US011537442B2

(12) **United States Patent**
Lippert et al.

(10) **Patent No.: US 11,537,442 B2**
(45) **Date of Patent: Dec. 27, 2022**

(54) **APPLICATION RUNTIME DETERMINED DYNAMICAL ALLOCATION OF HETEROGENEOUS COMPUTE RESOURCES**

(71) Applicant: **ParTec Cluster Competence Center GmbH**, Munich (DE)

(72) Inventors: **Thomas Lippert**, Aschaffenburg (DE); **Bernhard Frohwitter**, Munich (DE)

(73) Assignee: **ParTec Cluster Competence Center GmbH**, Munich (DE)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 35 days.

(21) Appl. No.: **16/963,749**

(22) PCT Filed: **Jan. 23, 2019**

(86) PCT No.: **PCT/EP2019/051615**
§ 371 (c)(1),
(2) Date: **Jul. 21, 2020**

(87) PCT Pub. No.: **WO2019/145354**
PCT Pub. Date: **Aug. 1, 2019**

(65) **Prior Publication Data**

US 2021/0096923 A1 Apr. 1, 2021

(30) **Foreign Application Priority Data**

Jan. 23, 2018 (EP) 18152903

(51) **Int. Cl.**
G06F 9/50 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 9/5061*** (2013.01); *G06F 2209/509* (2013.01)

(58) **Field of Classification Search**
CPC G06F 9/5061; G06F 2209/509
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

2012/0317168 A1* 12/2012 Driesen G06Q 30/06 709/202
2013/0282787 A1* 10/2013 Lippert H04L 29/08081 709/201

FOREIGN PATENT DOCUMENTS

EP 2442228 A1 4/2012
WO WO-2018065530 A1 4/2018

OTHER PUBLICATIONS

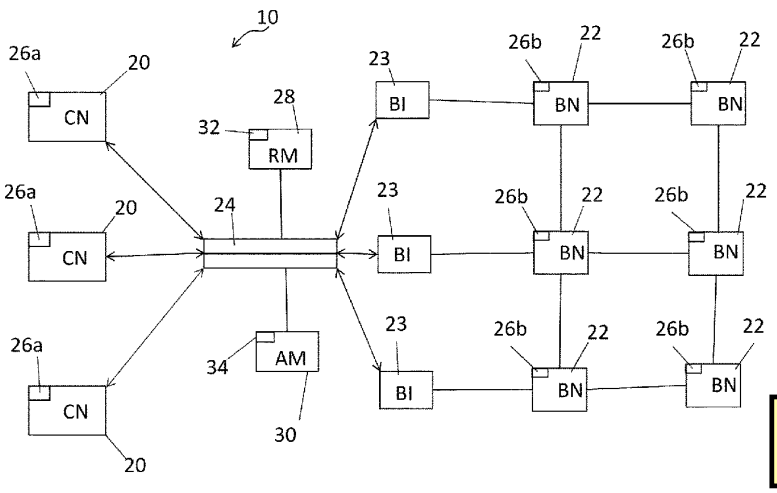
International Search Report for International Application No. PCT/EP2019/0518615, dated Apr. 29, 2019.
(Continued)

Primary Examiner — Sisley N Kim
(74) *Attorney, Agent, or Firm* — Marshall, Gerstein & Borun LLP

(57) **ABSTRACT**

The present invention provides a method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, wherein in a first computing iteration, the plurality of sub-tasks are assigned to and processed by ones of the plurality of computation nodes and booster nodes in a first distribution; and information relating to the processing of the plurality of sub-tasks by the plurality of computation nodes and booster nodes is used to generate a further distribution of the sub-tasks between the computation nodes and booster node for processing thereby in a further computing iteration.

10 Claims, 1 Drawing Sheet



Darell Long, Ph.D.
09/17/2025
Exhibit A
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1001

(56) **References Cited**

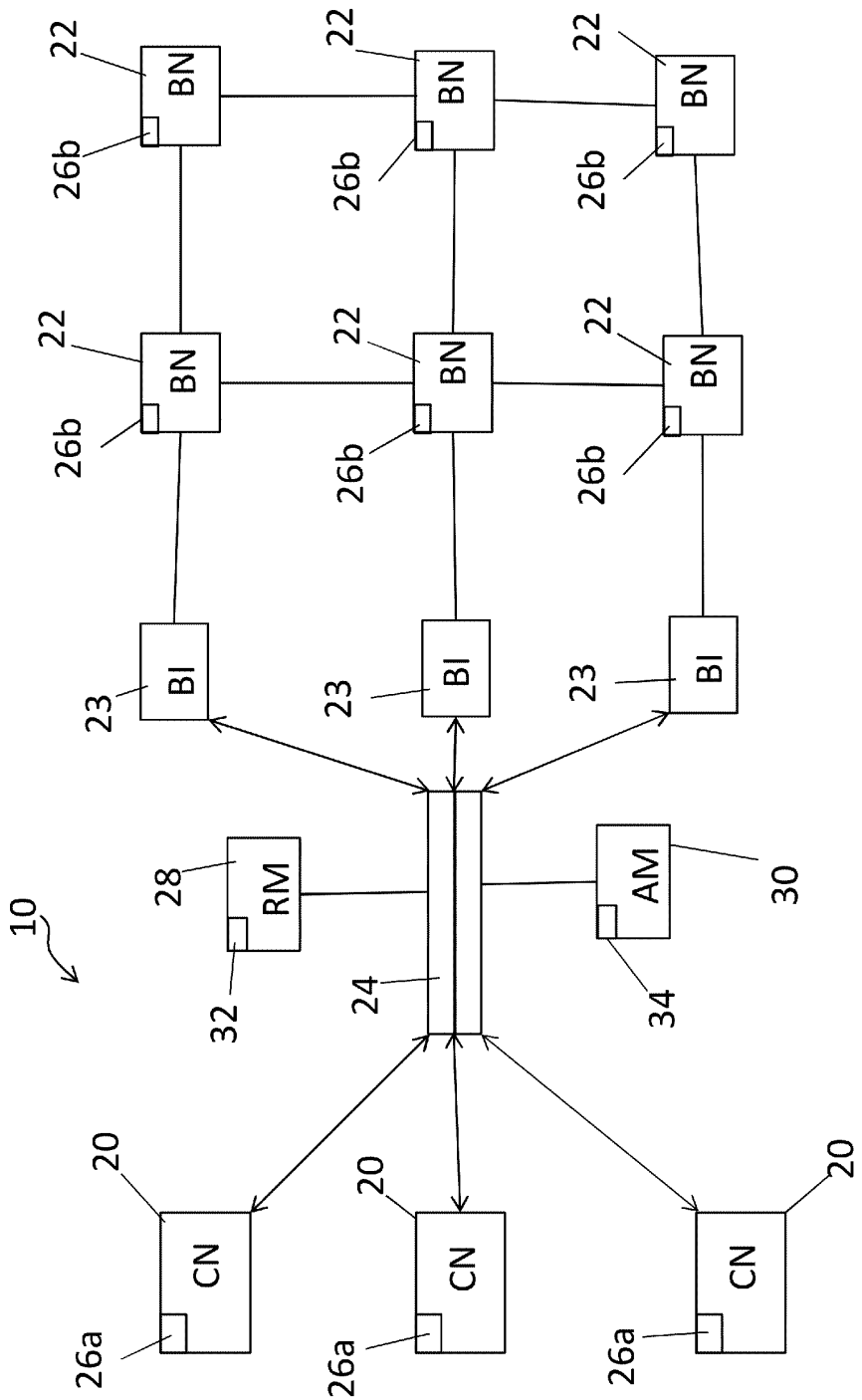
OTHER PUBLICATIONS

Written Opinion for International Application No. PCT/EP2019/0518615, dated Apr. 29, 2019.

"Dynamic Process Management With Allocation—Internal CO—Scheduling Towards Interactive Supercomputing," Clauss, et al., Conference Paper, DOI: 10.14459/2016md1288835, Jan. 19, 2016.

"The Deep Project an Alternative Approach to Heterogeneous Cluster-Computing in the Many-Core Era," Eicker, et al., Concurrency and Computation: Practice and Experience, Pract. Exper. 2016; 28; 2394-2411, Wiley Online Library (wileyonlinelibrary.com), Published online Jul. 27, 2015.

* cited by examiner



ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 3 of 411

APPLICATION RUNTIME DETERMINED DYNAMICAL ALLOCATION OF HETEROGENEOUS COMPUTE RESOURCES

CROSS-REFERENCE TO RELATED APPLICATIONS

This is the United States national phase of International Patent Application No. PCT/EP2019/051615, filed Jan. 23, 2019, which claims priority to EP 18152903.3 filed Jan. 23, 2018, the entire contents of each of which are hereby incorporated by reference herein.

FIELD OF THE DISCLOSURE

The present invention relates to a mechanism for executing computation tasks within a computation environment, in particular a heterogeneous computation environment adapted for parallel processing of a computation task.

BACKGROUND

The present invention is a development of the system described in earlier application WO 2012/049247 A1, WO '247, which describes a cluster computer architecture comprising a plurality of computation nodes and a plurality of boosters connected together via a communication interface. A resource manager is responsible for assigning one or more of the boosters and the computation nodes to each other in a dynamic manner during runtime. An example for such dynamic process management is described in Clauss et al. "Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing", COSH 2016 Jan. 19, Prague, CZ, incorporated herein by reference for all purposes. While the arrangement of WO '247 provides a flexible arrangement for assigning boosters to computation nodes, it does not address how to distribute tasks between computation nodes and boosters.

The arrangement of WO '247 is further described by Eicker et al. in "The DEEP Project An alternative approach to heterogeneous cluster-computing in the many core era", Concurrency Computat.: Pract. Exper. 2016; 28:2394-2411. The heterogeneous system which is described comprises a plurality of computation nodes and a plurality of booster nodes connected by a switchable network. In order to process an application, the application is "taskified" in order to provide an indication which tasks may be off-loaded from a computation node to a booster. This taskifying is achieved by the application developer annotating the code with pragmas indicating dependencies between different tasks and in addition labels indicating the highly scalable code parts which are to be processed by a booster. Scalability in this context means when the load offered to the service increases, an incremental and linear increase in hardware can maintain the same per-user level of service.

US 2017/0262319 A1 describes in one aspect runtime processes that can fully or partially automate the distribution of data and mapping of tasks to computing resources. A so-called tuning expert, i.e. a human operator, may still be required for mapping actions to available resources. In essence it is described how a given application to be computed can be mapped on a given computing hierarchy.

GENERAL DESCRIPTION

The present invention provides a method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, wherein in a first computing iteration, the plurality of sub-tasks are assigned to and processed by ones of the plurality of computation nodes and booster nodes in a first distribution; using information relating to the processing of the plurality of sub-tasks by the plurality of computation nodes and booster nodes to generate a further distribution of the sub-tasks between the computation nodes and booster nodes for processing thereby in a further computing iteration.

Preferably, the information is provided by respective daemons operating in each of the computation and booster nodes. The information provided allows an application manager to determine if the distribution of sub-tasks between computation nodes and booster nodes can be adapted or improved upon for a further computing iteration.

A resource manager may determine the assignment of tasks and sub-tasks to the computation nodes and booster nodes for the first iteration as a function of the computation task and further parameters. The application manager processes the information as input to the resource manager such that the resource manager dynamically alters further distributions during the computing of the computation task.

In a further aspect of the invention, the resource manager, based on the information dynamically alters assignment of the computation nodes and booster nodes to each other during the computation of the computation task.

The initial distribution may also be determined by the application manager using information provided by a user of the system in programming code compiled for execution by the system. Alternatively, the application manager may be so arranged to generate such a distribution based on an analysis of the sub-task coding.

In a further aspect, the invention provides a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks, a communication interface connecting the computation nodes with each other and the booster nodes, wherein the system comprises a resource manager for assigning booster nodes and computation nodes to each other for the computing of the tasks and wherein the system further comprises an application manager, the application manager being arranged to receive information from daemons operating in the computation nodes and booster nodes to update a distribution of the sub-tasks between the computation nodes and the booster nodes between a first computing iteration and a further computing iteration.

In still a further aspect of the invention, the resource manager receives the information such that the resource manager dynamically alters assignment of the computation nodes and booster nodes to each other during the computation of the computation task. Preferred embodiments of the invention will now be described, by way of example only, with reference to the accompanying drawings in which:

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. **1** shows a schematic diagram of a cluster computer system incorporating the invention.

DETAILED DESCRIPTION OF THE DRAWINGS

Referring to FIG. **1**, there is shown a schematic representation of a cluster computer system **10** incorporating the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 4 of 411

invention. The system **10** comprises a number of computation nodes **20** and a number of booster nodes **22**. The computation nodes **20** and the booster nodes are connected via a communication infrastructure **24**, with the booster nodes being connected to the communication interface via a booster interface **23**. Each of the computation nodes **20** and the booster nodes **22** is represented schematically by a rectangle and each of these nodes in operation incorporates at least one of a respective daemon **26***a* and **26***b*, schematically illustrated by a square within the rectangle of the respective node. A daemon of the present invention is an application which runs as a background process and is capable of providing the information as used herein. Clauss et al. "Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing", COSH 2016 Jan. 19, Prague, CZ, disclose daemons as addressed herein, and the contents of this paper are incorporated herein by reference for all purposes.

The system **10** also includes a resource manager **28** shown connected to the communication infrastructure **24** and an application manager **30**. Each of the resource manager **28** and the application manager **30** includes a respective daemon **32** and **34**.

The computation nodes **20** may be identical to one another or may have differing characteristics. Each computation node incorporates one or more multi-core processors such as the Intel XeonE5-2680 processor. The nodes are connected together by means of a communication interface which may be based on a Mellanox InfiniBand ConnectX fabric capable of transferring data at a rate of many Gbit/s. The computation nodes interface via the communication interface to the plurality of booster nodes, ideally through a series of booster interfaces **40**. As shown, the booster nodes host at least one accelerator-type processor, for example an Intel Xeon Phi many core processor, capable of autonomously booting and running its own operating system. Such technology is described in Concurrency Computat.: Pract. Exper. 2016; 28:2394-2411 referred to above.

Further, the system **10** may include a modular computing abstraction layer as described in unpublished application PCT/EP2017/075375, incorporated herein by reference for all purposes, for enabling communication between the daemons and the application manager.

A job to be computed by the system may comprise a number of tasks some of which or all may be repeated a number of times during the execution of the job. For example the job may be a "Monte-Carlo" based simulation where an effect is modelled using a random number, the calculations being repeated many times in succession.

The tasks may comprise a number of sub-tasks or kernels. Each of these sub-tasks may be more or less suitable for processing by one or more of the computation nodes or one or more of the boosters. In particular, the scalability of the sub-task may indicate whether it is more appropriate for the sub-task to be processed by a computation node or a booster. The system is flexible in all directions, also permitting joint processing of sub-tasks by all nodes addressed herein, as well as reshuffling of processing between the nodes.

If a task is computed with a first division of sub-tasks between a computation node and a booster, it is possible that such a division is not an optimal division for the computation of the task. Certain sub-tasks which were assigned to a booster on the first iteration might in fact not be suitable for processing by the booster such that a processing of the sub-tasks by a computation node rather than a booster might optimize the computation of the task as a whole. Accordingly, for a second and possible further iterations, if needed, of the task with an altered second and/or further sub-task distribution might improve the efficiency of the computation of the task.

Accordingly, the system **10** includes a mechanism whereby each of the computation nodes and the cluster nodes are arranged such that daemons **26***a*, **26***b* and **32** feed back information to the daemon **34** relating to the processing of sub-tasks and a current state of the respective processing entity. The daemon **34** uses the information provided by the daemons **26***a*, **26***b* and **32** to determine whether the distribution of sub-tasks to computation nodes and boosters could be adjusted to optimize or adapt the computation of the task for a subsequent iteration. The resource manager can also reassign computation nodes and boosters to each other, in addition to the adjusted distribution of tasks.

A job including a task in which an operator has estimated a scalability factor for each sub-task is input into the system. This task is compiled and the compiled code is executed. In the execution, a task is analyzed by the application manager and sub-tasks of the task are divided into computation node suitable sub-tasks and booster suitable sub-tasks and this information is passed to the resource manager for assignment of boosters to computation nodes. During a first iteration of the task, the results of the execution of the sub-tasks are collected together with the information from the daemons concerning the processing of the sub-tasks and the status of the nodes. The application manager then performs a reassignment of the sub-tasks for a subsequent iteration of the task and passes this updated assignment information to the resource manager which may also adjust the assignment of boosters to nodes accordingly.

For each iteration, the daemons operating in the computation nodes and the boosters report status information to the application manager and the resource manager enabling the calculation of subsequent iterations to be optimized by a further adjustments to the allocation of sub-tasks to computation nodes and boosters.

While the above procedure has been described incorporating a taskifying step in which an initial scalability factor may be inputted by the program coder, it is also possible for the application manager to automatically set an initial scalability factor for the sub-tasks, with subsequent iterations improving on this initial setting. Such an arrangement has the benefit that coding of the tasks is made more straightforward, thereby enabling an improved usability of the system to program coders unfamiliar with cluster computing applications.

As well as adjusting the distribution of sub-tasks between computation nodes and boosters based on a scalability of the sub-task, the distribution may also be influenced by information learned about the processing of the sub-task and any need to call further sub-tasks during the processing. If a first sub-task being handled by a booster requires input from a second sub-task not being handled by the booster, this may lead to an interruption in the processing of the first sub-task. Accordingly, the daemon at the booster handling the first sub-task can report this situation to the application manager such that in a further iteration both the first and second sub-tasks are handled by the booster. Accordingly, the application manager is arranged using the information provided by the daemons operating in the computation nodes and the boosters to adjust groupings of sub-tasks for assignment to the computation nodes and boosters.

While the computation nodes in FIG. **1** have been given the same reference numeral, as have the booster nodes, this does not imply that all the computation nodes are identical to each other and all the booster nodes are also identical to

each other. The system **10** may have computation nodes and/or booster nodes added to the system which differ in characteristics from the other computation/booster nodes. Accordingly, particular ones of the computation nodes and/or booster nodes may be particularly suited to processing particular sub-tasks. The application manager takes this structural information into account and passes such assignment information to the resource manager to ensure that the sub-tasks are distributed in an optimum manner.

An important aspect of the invention comes with the recognition that a mapping, tailoring of computation tasks, and sub-tasks onto a computer hierarchy may not provide sufficient use of the inherent flexibility and adaptability of the computer architecture, as shown as an example according to WO 2012/049247 A1. Therefore, the present invention integrates tailoring the application task most sufficiently, as an example according to WO 2017/0262319 A1, as well as configuring dynamically the computation nodes and booster nodes to each other, and eventually dynamically rearranging the mapping of the computation tasks during runtime, based on the information provided by daemons about the efficiency of the execution o the computation task, and dynamically reassigning the computation nodes and computation boosters to each other.

The invention claimed is:

1. A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:

in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and

generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

2. The method according to claim **1**, wherein an application manager receives the information and determines the further distribution.

3. The method according to claim **2**, wherein a resource manager determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task and wherein the application manager receives the information and processes the information as input to the resource manager such that the resource manager dynamically alters further distribution during the computing of the computation task.

4. The method according to claim **3**, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

5. The method according to claim **1**, wherein daemons operate in the plurality of computation nodes and the plurality of booster nodes to generate the information.

6. The method according to claim **1**, wherein the first distribution is determined based on a rating provided in source code for each sub-task in the plurality of sub-tasks.

7. The method according to claim **1**, wherein the information is used to provide a grouping of sub-tasks in at least one of the first computing iteration and the further computing iteration.

8. The method according to claim **3**, wherein a daemon operating at a node generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks.

9. A heterogeneous computing system comprising:

a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks;

a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes;

a resource manager for assigning at least a portion of the plurality of booster nodes and at least a portion of the plurality of computation nodes to each other for the computing of the one or more tasks in a first computing iteration; and

an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

10. The heterogeneous computing system according to claim **9**, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other.

* * * * *

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 6 of 411

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner,

v.

PARTEC CLUSTER COMPETENCE CENTER GMBH,
Patent Owner.

IPR2025-00318
U.S. Patent No. 11,537,442
Issued: December 27, 2022
Application No. 16/963,749
Filed: January 23, 2019

Title: Application Runtime Determined Dynamical Allocation of Heterogeneous Compute Resources

---

DECLARATION OF DARRELL LONG

Darell Long, Ph.D.
09/17/2025
Exhibit B
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1003

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 7 of 411

TABLE OF CONTENTS

Page(s)

I. INTRODUCTION AND ENGAGEMENT 4

II. BACKGROUND AND QUALIFICATIONS....4

III. MATERIALS CONSIDERED AND INFORMATION RELIED UPON REGARDING '442 PATENT 7

IV. UNDERSTANDING OF PATENT LAW 9

V. SUMMARY OF OPINIONS 13

VI. STATE OF THE ART....13

VII. LEVEL OF ORDINARY SKILL IN THE ART....17

A. Publications Reflecting The Knowledge Of A POSITA 19

VIII. THE '442 PATENT....21

A. The '442 Patent's Specification 21

B. The Prosecution History....23

IX. CLAIM CONSTRUCTION 24

X. CLAIMS 1-10 ARE OBVIOUS IN VIEW OF LIPPERT AND BUDENSKE 24

A. Lippert (EX1004) 24

B. Budenske (EX1005) 26

C. Combining Lippert And Budenske 27

1. Claim 1 36

2. Claim 2 45

3. Claim 3 47

4. Claim 4 53

5. Claim 5 53

6. Claim 6 55

7. Claim 7 59

8. Claim 8 60

9. Claim 9 61

10. Claim 10 64

XI. CLAIMS 2-5 AND 8-10 ARE OBVIOUS IN VIEW OF LIPPERT, BUDENSKE, AND KAMBATLA 64

A. Kambatla (EX1006) 65

B. Combining Lippert, Budenske, And Kambatla 67

1. Claim 2 73

2. Claim 3 76

3. Claim 4 79

4. Claim 5 79

5. Claim 8 81

6. Claim 9 82

7. Claim 10 84

I, Darrell Long, do hereby declare as follows:

I. INTRODUCTION AND ENGAGEMENT

1. I have been retained as an independent expert on behalf of Microsoft Corp. in connection with the above-captioned Petition for *Inter Partes* Review ("IPR") to provide my analyses and opinions on specific technical issues related to U.S. Patent No. 11,537,442 (hereinafter "the '442 patent").

2. I am being compensated at my usual and customary rate for my time with this IPR. My compensation is not affected by the outcome of this IPR.

3. Specifically, I have been asked to provide my opinions regarding whether claims 1-10 (each a "Challenged Claim" and collectively the "Challenged Claims") of the '442 patent would have been obvious to a person having ordinary skill in the art ("POSITA") as of January 2018. It is my opinion that claims 1-10 would have been obvious to a POSITA after reviewing the prior art discussed herein.

II. BACKGROUND AND QUALIFICATIONS

4. I am an expert in the field of distributed computing systems and the fields of computer science and computer engineering, more generally. For over 40 years, I have been working, teaching, or conducting research in the field of computer science and engineering. In formulating my opinions, I have relied upon my training, knowledge, and experience in the relevant art. A copy of my curriculum vitae is appended to this declaration as Appendix A and describes my professional

experience, including my academic and employment history, publications, conference participation, awards and honors, and more. The following paragraphs briefly summarize my relevant qualifications and professional experience.

5. I am a Distinguished Professor of Engineering, *Emeritus,* and a Distinguished Research Professor of Engineering at the University of California, Santa Cruz. I am also currently a Distinguished Visting Scholar at Santa Clara University. I received my B.S. degree in Computer Science from San Diego State University in 1984 and my M.S. and Ph.D. in Computer Science from the University of California, San Diego, respectively, in 1986 and 1988. I joined the faculty of the University of California, Santa Cruz, as an Assistant Professor of Computer Science in 1988. I earned tenure in 1994 and advanced to the rank of Associate Professor and in 1999 to the rank of full Professor, culminating in 2017 as Distinguished Professor of Engineering. I also twice served as Associate Dean for graduate studies and research and the Director of an NSF-funded research center.

6. In addition, I have been Professeur Invité at the Conservatoire National des Arts et Métiers, the Université Paris–Dauphine and Université Paris–Descartes and Sorbonne Université in Paris, France. I am Professor *ad Honorem* de la Universidad Católica del Uruguay and a visiting professor at the University of Technology, Sydney, Australia, and the United States Naval Postgraduate School. I am a European Organization for Nuclear Research (CERN) Associate Member.

7. Throughout my career, I have been actively involved in research. I have received over 50 peer-reviewed grants to fund my computer science and engineering research. I have authored two books and four chapters of books. I have also written 32 journal articles, 155 peer-reviewed conference papers, and 33 peer-reviewed short papers in computer science, including many papers related to distributed computing.

8. I have taught undergraduate and graduate courses on programming and programming languages, operating systems, storage systems, computer systems, cryptography and computer security, data structures, compiler design, and computer networks.

9. In addition to my academic positions, I have served on several special and standing committees of the National Research Council and the Science & Technology Committee for the Livermore and Los Alamos National Laboratories for over a decade. I served on the advisory board in Computer and Information Sciences for Sandia National Laboratory and the Global Security advisory board for Pacific Northwest National Laboratory. I was the Editor-in-Chief emeritus of Letters of the IEEE Computer Society, and am the Editor-in-Chief emeritus of the journal ACM Transactions on Storage, a Fellow of the Institute of Electrical and Electronics Engineers ("IEEE"), and a Fellow of the American Association for the Advancement of Science ("AAAS").

10. I am a named inventor on 11 U.S. Patents within the field of computer science, with a twelfth patent now being prosecuted.

11. I was a general chair for the conference on Mass Storage Systems and Technologies in 2024. I have been on multiple program committees and steering committees, including committees for the Symposium on File and Storage Systems Technology (a conference that I founded in 2002), the International Conference on Cloud Computing and Services Science, Symposium on Modeling, IEEE, International Conference on Cloud and Big Data Computing, Petascale Data Storage Workshop, and International Conference on Advanced Communications and Computation among many others.

III. MATERIALS CONSIDERED AND INFORMATION RELIED UPON REGARDING '442 PATENT

12. In preparing this declaration and arriving at my opinions, I have reviewed the following materials bearing Exhibit Nos. that I understand are being referenced in an IPR proceeding where my declaration may be referenced:

LIST OF EXHIBITS

No.	Description
1001	U.S. Patent No. 11,537,442 ("the '442 patent")
1002	File History of U.S. Patent No. 11,537,442
1003	N/A
1004	U.S. Patent App. Pub. No. U.S. 2013/0282787 ("**Lippert**")

No.	Description
1005	John R. Budenske et al., *A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms*, The Journal of Supercomputing, Vol. 12, No. 4 (Oct. 1998) ("**Budenske**")
1006	U.S. Patent App. Pub. No. U.S. 2018/0074855 ("**Kambatla**")
1007	Yu-Kwong Kwok et al., *A Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems*, Journal of Parallel & Distributed Computing (2005) ("**Kwok**")
1008	Mitchell D. Theys et al., *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach*, *in* Solutions to Parallel and Distributed Computing Problems (A. Zomaya, F. Ercal & S. Olariu eds., 2001) ("**Theys**")
1009	U.S. Patent No. 3,496,551 ("**Driscoll**")
1010	Yuan-Chieh Chow and Walter H. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System*, IEEE Transactions on Computers, Vol. C-28, No. 5 (May 1979) ("**Chow**")
1011	David M. Nicol and Paul F. Reynolds, Jr., *Optimal Dynamic Remapping of Data Parallel Computations*, IEEE Transactions on Computers, Vol. 39, No. 2 (Feb. 1990) ("**Nicol**")
1012	Arun C. Murthy et al., Apache Hadoop YARN (2014) ("**Murthy**")
1013	Morris Jette and Mark Grondona, *SLURM: Simple Linux Utility for Resource Management*, ClusterWorld Conference and Expo (2003) ("**Jette**")
1014	Eicker et al., *The DEEP Project: An Alternative Approach to Heterogeneous Cluster-Computing in the Many-Core Era*, Concurrency and Computation: Practice and Experience (2016) ("**Eicker**")

No.	Description
1015	U.S. Patent App. Pub. No. U.S. 2017/0262319 ("**Newburn**")
1016	Carsten Clauss et al., *Dynamic Process Management with Allocation-Internal Co-Scheduling Towards Interactive Supercomputing*, Proceedings of the First Workshop on Co-Scheduling of HPC Applications (2016) ("**Clauss**")
1017	N/A
1018	Samik Raychaudhuri, *Introduction to Monte Carlo Simulation*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Raychaudhuri**")
1019	David W. Bauer, Jr. and Mojdeh Mohtashemi, *An Application of Parallel Monte Carlo Modeling for Real-Time Disease Surveillance*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Bauer**")
1020	List of Challenged Claims
1021	U.S. Patent No. 5,881,227 ("**Brenner**")
1022	U.S. Patent No. 7,047,299 ("**Curtis**")
1023	N. Metropolis, *The Beginning of the Monte Carlo Method*, Los Alamos Science, Special Issue 125-130 (1987) ("**Metropolis**")

IV. UNDERSTANDING OF PATENT LAW

13. I am not an attorney. For this declaration, I have been informed about certain aspects of the law that are relevant to my opinions. The Petitioner's attorneys provided me with my understanding of the law.

14. I understand that when considering the scope of the claims of a patent, the patent claim terms should generally be given the ordinary meaning that the terms

would have to a person of ordinary skill in the art in question after reading the patent as of the earliest claimed priority date.

15. I understand that the person of ordinary skill in the art is deemed to read the claim term not only in the context of the particular claim in which the term appears but in the context of the entire patent, including the other claims and the application specification as filed. I further understand that the principal considerations regarding the claims' scope and meaning are the claims' plain language (including the surrounding claim language and context), the patent specification, and the patent's prosecution history in the Patent Office. I understand that while a claim is to be read in light of the specification, one must generally avoid importing limitations into the claim from the specification. I am also informed that the prosecution history can often inform the meaning of the claim by demonstrating how the inventor understood the invention and whether the inventor limited the invention in the course of prosecution, making the claim scope narrower than it would otherwise be. I applied these understandings when considering the scope and meaning of the claims of the '442 patent.

16. I further understand that a claim is unpatentable if it would have been obvious. Obviousness of a claim requires that the claim would have been obvious from the perspective of a POSITA at the time the alleged invention was made. I understand that a claim could have been obvious from a single prior art reference or

from a combination of two or more prior art references. I understand that an obviousness analysis requires understanding the scope and content of the prior art, any differences between the alleged invention and the prior art, and the level of ordinary skill in evaluating the pertinent art.

17. I further understand that a claim would have been obvious if it unites old elements with no change to their respective functions, or merely substitutes one element for another known in the field, and that combination yields predictable results. While it may be helpful to identify a reason for this combination, I understand there is no strict requirement to find an express teaching, suggestion, or motivation to combine within the references. When a product is available, design incentives and other market forces can prompt variations of it, either in the same field or a different one. If a POSITA can implement a predictable variation, obviousness likely prevents its patentability. For the same reason, if a technique has been used to improve one device and a POSITA would recognize that it would improve similar devices in the same way, using the technique would have been obvious. I understand that a claim would have been obvious if common sense directs one of ordinary skill, without hindsight bias caused by knowledge of the challenged claims, to combine multiple prior art references or add missing features to reproduce the alleged invention recited in the claims.

18. I further understand that certain factors may support or rebut the obviousness of a claim. I understand that such secondary considerations include, among other things, commercial success of the patented invention, skepticism of those having ordinary skill in the art at the time of invention, unexpected results of the invention, any long-felt but unsolved need in the art that was satisfied by the alleged invention, the failure of others to make the alleged invention, praise of the alleged invention by those having ordinary skill or expertise in the art, and copying of the alleged invention by others in the field. I understand that there must be a nexus—a connection—between any such secondary considerations and the alleged invention. I also understand that contemporaneous and independent invention by others is a secondary consideration tending to show obviousness.

19. I am not aware of any allegations by the named inventors of the ’442 patent or any assignee of the ’442 patent that any secondary considerations would rebut the obviousness of any Challenged Claim of the ’442 patent.

20. I understand that in considering obviousness, it is important not to determine obviousness using the benefit of hindsight derived from the descriptions found in the patent being considered.

21. I understand that Petitioner has the burden of proving obviousness by a preponderance of evidence, which means that the claims are more likely than not obvious.

22. The analysis in this declaration is in accordance with the above-stated legal principles.

V. SUMMARY OF OPINIONS

23. In forming the opinions set forth herein, I have considered:

a. The documents listed above;

b. My education, knowledge, skills, and experience in the field of computer science and engineering; and

c. The level of skill of a person having ordinary skill in the art (POSITA) at the time of the alleged invention of the ’442 patent, namely January 23, 2018.

24. As I explain in further detail below, it is my professional and expert opinion that claims 1-10 of the ’442 patent are obvious over the prior art.

VI. STATE OF THE ART

25. A heterogeneous computing system is a computer architecture that uses multiple types of processors, each better at performing certain particular tasks, and operating systems. *E.g.*, EX1008, 135. POSITAs have long known how to assign tasks between multiple processors for multi-processor systems and heterogeneous computing systems. *E.g.*, EX1009, 1:70-2:4 (as of 1967, “the attention of systems designers is turning ever increasingly towards the concept of multi-processing computing systems wherein a particular large job may be broken up into individual

segments or tasks and performed in parallel in two or more separate processors wherever possible"); EX1010, 354 (describing scheduling tasks to dynamically load balance a "heterogeneous multiple processor system" in 1979).

26. A well-known "design goal of any multi-processor computing system is to achieve high overall efficiency." EX1009, 2:13-20. To achieve overall efficiency throughout processing of a task, POSITAs have long known of benefits of assigning tasks to processors *dynamically* because "[s]ome computations dynamically change their behavior as they progress." EX1011, 206.

27. One known type of distributed computing architecture is a "cluster," which is a collection of interconnected computing devices ("nodes") utilized as a single, unified computing resource. EX1022, 1:16-54 (describing clusters). Clusters often use "daemons," a daemon being a background process that performs a specific task[1]. EX1021, 7:66-67 (a "daemon" is defined as "a program that runs unattended to perform a standard service"), 8:2-3 ("Daemon processes generally provide services that must be available at all times."), 2:54-3:18 (describing using daemons to facilitate communication in a "parallel processing environment" comprising "a

[1] The term "daemon" is commonly used in computer science to refer to background processes, inspired by Maxwell's daemon of physics and thermodynamics. *See* https://ei.cs.vt.edu/~history/Daemon.html.

plurality of nodes connected through a network to one another"); EX1022, 3:63-66 ("As is commonly known, a daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events."), 3:66-4:14 (explaining that "[t]ypical daemon processes … perform administrative tasks" and are implemented in "any environment … which is capable of implementing background processes and supports communications between the various Nodes of the Cluster(s)"). As of the earliest alleged filing date, POSITAs typically used cluster resource management frameworks such as Hadoop YARN or SLURM to manage the allocation of jobs to cluster resources. EX1012, xvii (explaining that Hadoop is "a compelling part of a modern data architecture for a very wide spectrum of the industry"), xviii (explaining that "YARN makes the effort of building such [distributed] systems significantly simpler by dealing with many issues with which a framework developer would be confronted"), 1-20 (providing a historical overview of Hadoop YARN), 43-58 (providing a "Functional Overview of YARN Components" for managing cluster resources); EX1013, 3 (explaining that "Simple Linux Utility for Resource Management (SLURM) is a resource management system suitable for use on large and small Linux clusters"), 4 (explaining that "[a]s a cluster resource manager, SLURM … allocates exclusive and/or non-exclusive access to resources (compute nodes) to users" and "provides a framework for starting, executing, and monitoring work (normally a parallel job) on

the set of allocated nodes"). Such frameworks may include a daemon at every cluster node to enable monitoring and control of the node by a central resource management daemon. EX1012, 9 (describing a central daemon and node daemons or "TaskTrackers" for executing tasks at the node), 38-40 (describing a "NodeManager" daemon at each node "responsible for launching the applications' containers, monitoring their resource usage …, and reporting the same to the ResourceManager"); EX1013, 5-7 (describing a central controller daemon and daemons running at each compute node "to export control to SLURM"), 20 (describing a daemon at each node "for managing user jobs and monitoring system state," where its "most common action … is to report system state on request").

28. The '442 patent includes various acknowledgments regarding the state of the art. For example, the '442 patent acknowledges that heterogeneous computing systems comprising a plurality of computation nodes and a plurality of booster nodes were known. EX1001, 1:23-27. The '442 patent acknowledges that prior art reference Eicker (EX1014), describes such a heterogeneous computing system capable of "taskifying" an application into a plurality of tasks that are distributed among the nodes to enable scalability. EX1001, 1:39-55. The '442 patent acknowledges that Newburn (EX1015) describes "runtime processes that can fully or partially automate the distribution of data and mapping of tasks to computing resources." EX1001, 1:56-58; *see also* EX1001, 5:10-24. The '442 patent further

acknowledges that resource managers for "dynamic process management" were known and "described in Clauss" (EX1016), a prior art reference which also discloses "daemons as addressed" in the '442 patent. EX1001, 1:28-35, 3:13-17. As an example of a job with tasks that may be repeated during execution, the '442 patent suggests a "'Monte-Carlo' based simulation," a well-known method in the art that was also known for efficient processing with cluster computing systems. EX1001, 3:44-49; EX1018, 91 (providing an "introductory tutorial on Monte Carlo simulation, a type of simulation that relies on repeated random sampling and statistical analysis to compute the results"); EX1019, 1029-1031 (describing a cloud computing architecture for "opportunistic" scheduling of "parallel monte carlo modeling methods" among compute nodes of the system), 1032-1036 (discussing scalability of Monte Carlo methods and demonstrating the "highly efficient parallel computation for monte carlo models"); EX1023, 125-130 (discussing the development of the Monte Carlo method for early computers in the 1940s, and the state of parallel processing in 1987 for using the method).

VII. LEVEL OF ORDINARY SKILL IN THE ART

29. I understand that in evaluating obviousness of the '442 patent claims, the content of a patent or printed publication prior art should be interpreted the way a person of ordinary skill in the art would have interpreted the prior art as of the effective filing date of the challenged patent claims. I assume that the effective filing

date is the date of the European patent application to which the '442 patent alleges priority, namely January 23, 2018. My understanding of a POSITA would not materially differ if the effective filing date were the PCT filing date or the U.S. application date, namely January 23, 2019, or July 21, 2020.

30. I understand that factors that may be considered in determining the level of ordinary skill in the art at the time of the effective filing date of the challenged patent include: (1) the educational level of the inventor; (2) the types of problems encountered in the art; (3) prior art solutions to those problems; (4) rapidity with which innovations are made; (5) sophistication of the technology; and (6) educational level of active workers in the field.

31. A person of ordinary skill in the art ("POSITA") would include someone who had, in January 2018, (i) a bachelor's degree in computer science, electrical engineering, computer engineering, or a closely related field, as well as (ii) approximately three years of industry experience or research with distributed computing technologies. Additional education could substitute for the industry experience as it typically involves at least 2-4 years of additional study focused exclusively on computer science, as well as a thesis or project spanning several terms to implement a substantial project.

32. Such industry experience working with distributed computing technologies would have included experience designing, implementing, and

maintaining distributed computing systems, such as cluster computing systems, installing and adapting cluster resource management frameworks on distributed computing systems, and deploying applications on such systems, as described above in the State Of The Art. A POSITA would have known of the basic design principles of fault tolerance and efficient scheduling/task assignments.

33. I would have met the requirements of a POSITA in the period from January 23, 2018, to July 21, 2020. Some of my past students would also have qualified as POSITAs on these dates.

A. Publications Reflecting The Knowledge Of A POSITA

34. The knowledge of a POSITA is reflected in well-known and widely available books and industry publications, which confirm my own opinions as to the knowledge of a POSITA. These books and industry publications, described below, would have been known and readily available to POSITAs in January 2018.

35. Budenske (EX1005) published in 1998 in the well-regarded Journal of Supercomputing, which has been publishing papers on the technology and applications of supercomputing since 1987. Budenske describes dynamically re-mapping sub-tasks among processors of a heterogeneous computing system between iterations in order to minimize the execution time of each iteration.

36. Kwok (EX1007) published in 2005 by Elsevier in the Journal of Parallel and Distributed Computing expands the Budenske methodology by evaluating

techniques to associate dynamic parameters with mappings. Theys (EX1008) published in 2001 by Wiley, a well-known publisher, as part of the book *Solutions to Parallel and Distributed Computing Problems*, which covers biology-inspired techniques for computing (e.g., neural networks, genetic algorithms). Theys describes the Budenske methodology, with a focus on using genetic algorithms to map sub-tasks to processors optimally.

37. Chow (EX1010) and Nicol (EX1011) were both published in the journal IEEE Transactions on Computers, a well-respected journal widely known in the art, in 1979 and 1990 respectively.

38. Murthy (EX1012) was published in 2014 and is a well-known text on Apache Hadoop YARN written by its creators that describes how to install and administer clusters with YARN, as well as how to develop applications and frameworks with YARN.

39. Jette (EX1013) published in 2003 and is an overview of SLURM, the "open source, fault-tolerant, and highly scalable cluster management and job scheduling system for Linux clusters of thousands of nodes." EX1013, 3.

40. Eicker (EX1014) published on June 10, 2016, by Wiley InterScience in the journal Concurrency and Computation: Practice and Experience.

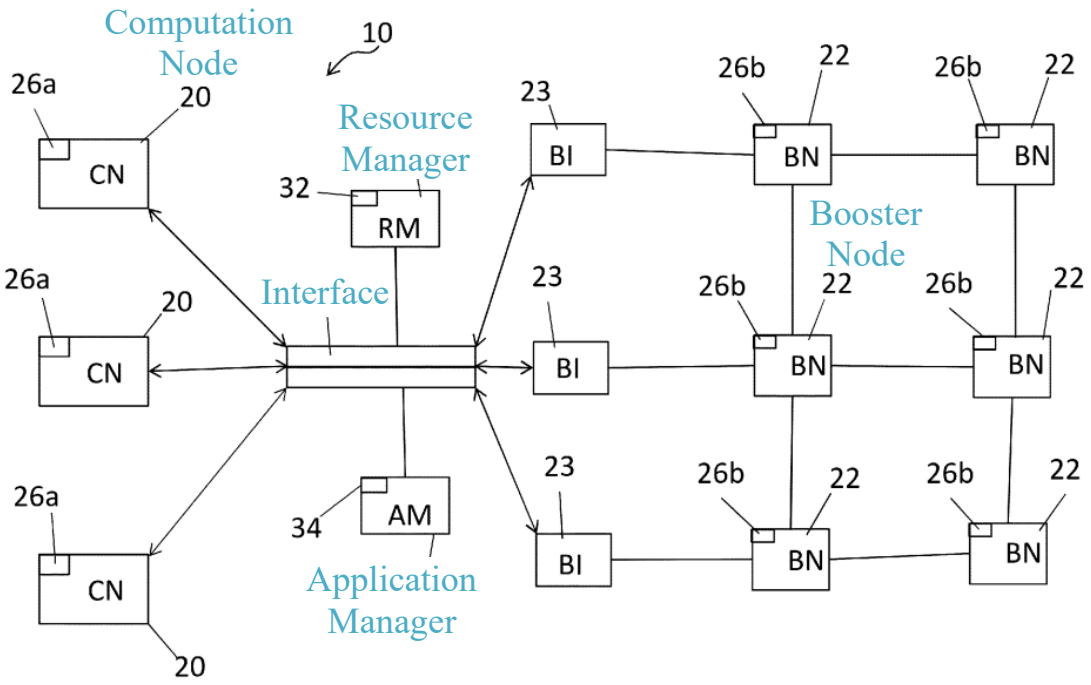
41. Clauss (EX1016) was presented and published on January 19, 2016, in the First Workshop on Co-Scheduling of HPC Applications in Prague, Czech Republic. EX1016, 1.

42. Raychaudhuri (EX1018) and Bauer (EX1019) were presented at the 2008 Winter Simulation Conference in Miami, Florida, and published in the Proceedings of the 2008 Winter Simulation Conference. *E.g.*, EX1018, i, 91; EX1019, i, 1029.

VIII. THE '442 PATENT

A. The '442 Patent's Specification

43. The '442 patent describes a "cluster computing system 10" comprising "a number of computation nodes 20 and a number of booster nodes 22" that are "arranged to compute a computation task … comprising a plurality of sub-tasks." EX1001, 1:66-2:5, 2:66-3:2. A computation node includes a "multi-core processor," while a booster node includes an "accelerator-type processor." *Id.*, 3:25-27, 3:33-36. The '442 patent explains that these devices and architecture were known in the art, with booster nodes known and used for processing "highly scalable code parts" or sub-tasks of a computation task. *Id.*, 1:23-55, 3:25-38, 3:53-55.



EX1001, FIG. 1 (annotated). The cluster includes a resource manager and an application manager, as well as daemons associated with each element of the cluster where each daemon "runs as a background process" in the respective element. EX1001, 3:6-23. The application manager collects information from daemons at the nodes and provides this information to the resource manager "to determine if the distribution of sub-tasks between computation nodes and booster nodes can be adapted or improved upon for a further computing iteration." EX1001, 2:13-25. Although the resource manager and application manager are depicted as separate components, a computer scientist would not be concerned if it were one or more, or if one was a subcomponent of the other, because that is just a software structuring choice and does not bear at all on functionality.

44. According to the ’442 patent, “[a] job to be computed by the system may comprise a number of tasks some of which or all may be repeated a number of times during the execution of the job.” EX1001, 3:44-49. The ’442 patent briefly describes distributing sub-tasks between computation nodes and boosters, but that an initial distribution of sub-tasks between a computation node and a booster during a first iteration may not be optimal, so altering the distribution of the sub-tasks among nodes for later iterations “might improve the efficiency of the computation of the task.” EX1001, 3:59-4:3.

B. The Prosecution History

45. During the prosecution of the ’442 patent, the Patent Office Examiner initially rejected all claims as anticipated (fully disclosed) by Lippert (EX1004). EX1002, 155-166. In response, the Applicant argued that Lippert does not anticipate the claims because Lippert does not disclose processing the plurality of sub-tasks in “multiple computing iterations.” *Id.*, 174 (emphasis omitted). Applicant further argued that the claims were distinguished from Lippert because they “allow for redistribution based on information relating to the processing of the plurality of sub-tasks,” whereas according to Applicant, “Lippert merely proposes a load balancing based on the workload of the nodes.” *Id.*, 174-175.

46. The Examiner then allowed all pending claims. *Id.*, 184.

47. In my view, the Examiner should not have been persuaded by the Applicant's arguments, in part because iterative processing has been a basic feature of computing systems since their inception.

IX. CLAIM CONSTRUCTION

48. For this declaration, I apply the plain and ordinary meaning of all claim terms, and do not rely on any specific constructions. I reserve the right to offer my opinions on any constructions or construction issues presented by Patent Owner ParTec.

X. CLAIMS 1-10 ARE OBVIOUS IN VIEW OF LIPPERT AND BUDENSKE

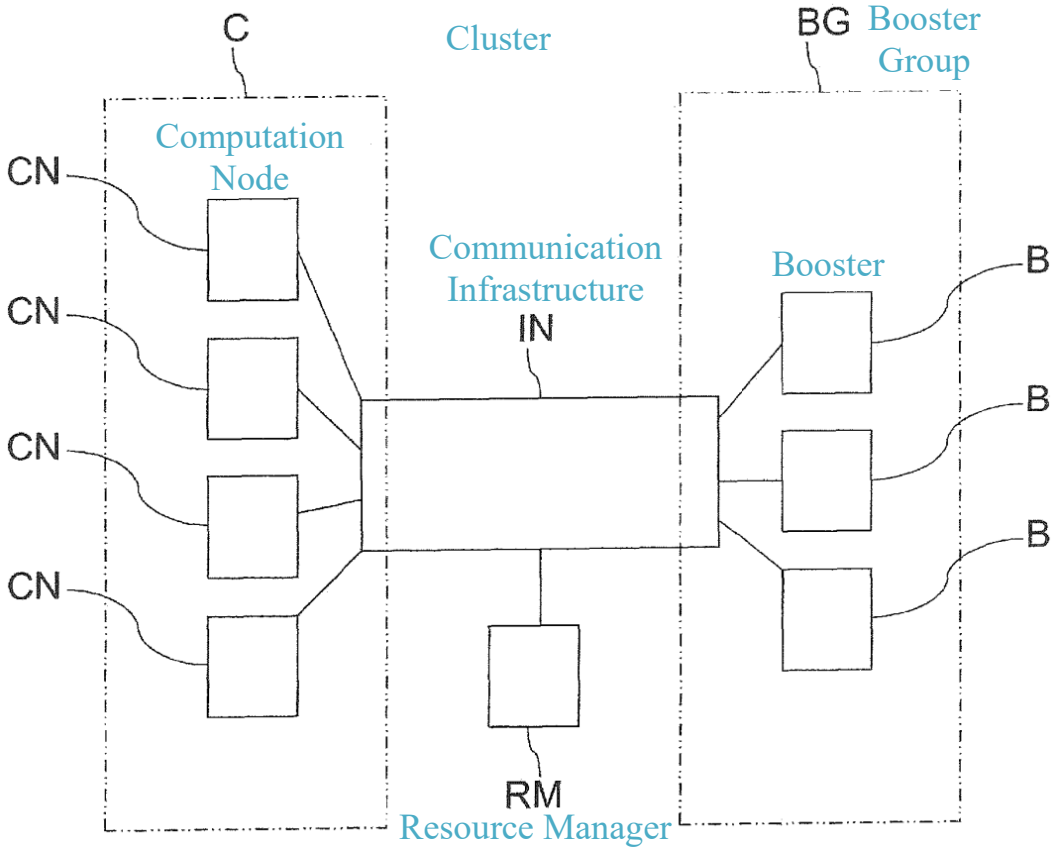
49. As explained below, claims 1-10 are obvious over Lippert in view of Budenske.

A. Lippert (EX1004)

50. Lippert is a U.S. patent application published on October 24, 2013. EX1004, Cover. Lippert is the same Thomas Lippert named as the first co-inventor on the '442 patent.

51. Lippert describes a "computer cluster arrangement for processing a computation task," the arrangement comprising a plurality of computation nodes and a plurality of boosters. EX1004, [0009]-[0011], FIG. 2. In Figure 2, Lippert depicts an example computer cluster with a plurality of computation nodes CN and a

plurality of boosters B in a booster group BG; the computation nodes CN and the boosters B are interconnected via a communication infrastructure IN. EX1004, [0067]. To process a computation task, a resource manager RM "initializes a first assignment and further on establishes a dynamic assignment of boosters B to computation nodes CN." EX1004, [0068], [0013].



EX1004, FIG. 2 (annotated).

52. The computation task is allocated among computation nodes and boosters in part based on the processing requirements of each part of the computation task and the processing capacities, where computation nodes compute the first part

of the computation task while a booster computes a second part of the computation task. EX1004, [0017], [0010]-[0012], [0014]-[0016], [0018]-[0021], [0027]-[0029].

B. Budenske (EX1005)

53. Budenske is a printed publication published October 1998 in Volume 12, Number 4 of the Journal of Supercomputing. EX1005, 387. The Journal of Supercomputing is a well-regarded journal in the art, and Budenske would have been readily available to a person interested in the art around the time of its publication date. I am particularly familiar with the Journal of Supercomputing, as I served as a reviewer for this journal in 2022. EX1003, Exhibit A.

54. Budenske teaches the execution of iterative tasks with a "heterogeneous parallel hardware platform," and the real-time re-mapping of sub-tasks to processors between iterations based on the sub-tasks' processing of input data in a preceding iteration. EX1005, 387. An "Intelligent Operating System" (IOS) "make[s] a heuristically-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data." *Id.*, 387-388. A central module of the IOS (HC Kernel or HC Kernel Monitor) (1) establishes an initial mapping of the sub-tasks onto the processing resources of the heterogeneous computing system, (2) monitors the execution of the sub-tasks, and (3) at the end of each iteration, decides whether and how to change the mapping of the sub-tasks onto the hardware based on information about dynamic

parameters obtained from processing the sub-tasks in the preceding iteration. EX1005, 401-402.

55. Budenske teaches determining a new mapping of sub-tasks to processing resources "in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration." EX1005, 402. Budenske teaches "reconfigur[ing] the assignment of processing resources to subtasks **to reduce execution time** of the next iteration." EX1005, 391 (emphasis added).

C. Combining Lippert And Budenske

56. A POSITA would have been motivated, with a reasonable expectation of success, to combine Lippert's teachings on a computer cluster arrangement with dynamic assignments of booster nodes to computation nodes with Budenske's teachings on re-mapping sub-tasks between iterations of an iterative application. A skilled artisan would have combined these respective teachings into an improved design (hereinafter "Lippert-Budenske") that included benefits from each reference.

57. The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
    - A plurality of computation nodes;
    - A plurality of booster nodes; and

- A communication infrastructure interfacing each of the computation nodes and the booster nodes;

- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-402); and
  - Dynamically assign, during the processing of the computation task, computation nodes, and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);

- An application manager module, as taught by Budenske (*e.g.*, EX1005, 388, 402), arranged to:
  - Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);

- Monitor execution of sub-tasks to collect information relating to the processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
- Assign, according to a new mapping obtained using information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-402).

58. A POSITA would have been motivated, with a reasonable expectation of success, to so combine the teachings of Lippert and Budenske for at least the following reasons.

59. *First*, Lippert and Budenske each disclose heterogeneous computing systems that process an application or computation task with dynamic assignments. *E.g.*, EX1004, [0015]; EX1005, 387. Lippert discloses a flexible assignment of boosters to computation nodes to facilitate scalability, where parts of a computation task are allocated between computation nodes and boosters. *E.g.*, EX1004, [0013], [0015]-[0020]. Budenske discloses a flexible assignment of sub-tasks of an iterative application to processors of a heterogeneous computing system so that each sub-task is assigned "where it is best suited for execution" for each iteration. EX1005, 390, 387. A POSITA would have been motivated to implement the heterogeneous computing system of Lippert with the dynamic, iterative sub-task assignment

method of Budenske partly because doing so would facilitate efficient distributions of sub-tasks and processing resources during execution of an iterative application, thus fully leveraging the flexibility of the heterogeneous architecture and "reduc[ing] execution time" of computing iterations. EX1005, 391.

60. *Second*, Budenske provides express motivation for implementing its method in a heterogeneous computing system such as Lippert, namely to "fully utilize the architectural flexibility of such a [heterogeneous parallel computing] system." EX1005, 390. A POSITA would have found that the dynamic flexibility of the heterogeneous computing system taught by Lippert through the "loose coupling" of computation nodes and boosters (EX1004, [0013], [0015] (describing "a maximum flexibility in providing hardware resources")), naturally complements the dynamic flexibility of remapping sub-tasks among processing resources to account for the actual processing of the sub-tasks by the processing resources during preceding iterations, as taught by Budenske (EX1005, 391), and that the combination of these teachings would provide increased flexibility for optimally processing iterative applications.

61. *Third*, Lippert recognizes that it has long been known to outsource portions of computation tasks "of high resource requirements" to accelerators (boosters), while other portions of the computation tasks are executed by computation nodes. EX1004, [0003], [0028]-[0029] (describing how computation

nodes and boosters are "typically" configured with processors suited for different tasks), [0068], [0084]-[0086]. Budenske similarly recognizes the varying requirements of different computations of a given computation task, and that "[h]eterogeneous parallel architectures are ideal computing platforms for efficiently handling computational tasks with such diverse requirements." EX1005, 390 (discussing a computation task that involves "numeric computation, quasi-symbolic computation …, and symbolic computation"). A POSITA would have recognized that Lippert and Budenske follow the same principles of efficient processing of a task, comprising a plurality of sub-tasks, with a heterogeneous computing system, by assigning sub-tasks to processors (e.g., computation nodes or boosters) best suited to execute the sub-task. A POSITA thus would have been motivated to implement Budenske's teachings with a heterogeneous computing system including boosters, as taught by Lippert, partly because "the boosters are implemented to process specific problems at high speed." EX1004, [0028].

62. *Fourth*, a POSITA reading Lippert would have understood the need to account for sub-task assignment among nodes for iterative applications, which Budenske teaches. While Lippert focuses on the configuration of the processing resources of the system itself, Lippert only broadly and generally discusses the nature of the "application" or "computation task" being computed. *E.g.*, EX1004, [0015]-[0016]. Further, Lippert discloses dynamically assigning boosters to

computation nodes during runtime, and outsourcing parts (i.e., sub-tasks) of the computation tasks to the assigned boosters, suggesting that sub-tasks are also dynamically remapped to newly assigned boosters during processing of the task and "in dependency of processing a specific computation task." EX1004, [0013] ("establish a dynamic assignment at runtime, which means during processing of the computation task"), [0015] ("booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"). Budenske, meanwhile, focuses on applications where sub-tasks are iteratively and concurrently executed, and where the processing requirements for a sub-task may evolve over time. EX1005, 387, 390. To that end, Budenske teaches monitoring dynamic parameters of the application during each iteration—where these dynamic parameters relate to the processing of the input data and "have the most impact on the execution time of the subtasks in the application"—and adjusting the assignment of sub-tasks for the next iteration accordingly based on the dynamic parameters. EX1005, 391, 393. A POSITA thus would have naturally turned to Budenske when implementing Lippert for iterative applications in order to "assign each … subtask to the processors where it is best suited for execution" and thus minimize execution time as the processing requirements of the computation task evolves over time. EX1005, 390.

63. *Fifth*, a POSITA would have recognized the compatibility of Lippert and Budenske's teachings. Lippert, for example, recognizes that a computation task can be divided into at least a first part for computing by computation nodes, and a second part for computation by boosters. EX1004, [0009]-[0014], [0016]-[0020], [0068]. Further, code blocks (e.g., sub-tasks) are classified as scalable or complex. EX1004, [0084]. A POSITA would have understood that the "first part of the computation task" computed by the computation nodes includes the less scalable, complex sub-tasks, while the "second part of the computation task" computed by the booster(s) includes the highly scalable sub-tasks. *Id.*; EX1014, 5. Further, Lippert recognizes that "[e]ach code has highly scalable and less scalable complex elements," and so a POSITA would have been motivated to change which sub-tasks are assigned to a first part (computed by computation nodes) and a second part (computed by boosters), regardless of a sub-task's initial classification as complex or scalable (and thus initial mapping to computation nodes or boosters), if information relating to the processing of the sub-task in a preceding iteration suggests that a different node would better process the sub-task.

64. Nothing in these references teaches away from this combination. On the contrary, a POSITA would have recognized that the heterogeneous computing systems of Budenske and Lippert are similar. Specifically, Budenske discloses a heterogeneous computing system including DSP (digital signal processors) and

RISC (reduced instruction set computer) processors (EX1005, 391), and a POSITA would have recognized that the DSPs are processors that function as accelerators/boosters and the RISC processors as computing nodes, much like the processing elements in Lippert. A POSITA thus would have understood that the teachings of Lippert and Budenske are compatible and would work well together.

65. *Finally*, a POSITA would have reasonably expected to succeed in implementing the Lippert-Budenske combination because doing so would only entail providing software code, which was well within the skill of a POSITA. Further, a POSITA would have reasonably expected success implementing the teachings of Budenske in a heterogeneous computing system as taught by Lippert partly because there is nothing incompatible between the two approaches, and because Budenske explains that its teachings "can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the application and platforms considered here." EX1005, 404. Other prior art references available to the POSITA disclosed successful implementations of Budenske's teachings and confirmed that Budenske's approach to re-mapping sub-tasks is feasible and achieves faster execution times compared to other dynamic task scheduling techniques. EX1007, 78-79 (describing Budenske), 92 (illustrating execution times of Budenske's "On-Off" approach compared to other dynamic scheduling approaches for task graphs of various sizes and types); EX1008, 156-165

(discussing Budenske's approach and performance results). In view of these references, a POSITA would have reasonably expected to succeed implementing Lippert-Budenske and achieving the benefits described in each reference.

66. Further, the '442 patent provides only a high-level description of adjusting the distribution of sub-tasks between computation nodes and boosters. *E.g.*, EX1001, 3:59-4:3, 4:23-31. This level of disclosure reflects the state of the art, as it assumes anyone desiring to carry out the process would know of the equipment and techniques to be used, none being specifically described, and thus confirms that POSITAs would have reasonably expected to succeed in implementing Lippert-Budenske.

67. So combined, the Lippert-Budenske combination dynamically distributes sub-tasks of a computation task among computation nodes and boosters of a heterogeneous computation system between iterations of the computation task, to more efficiently process the computation task, and thus renders all challenged claims obvious.

1. Claim 1

a) Element [1.1] [2]

[1.1] A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:

68. Lippert-Budenske discloses and teaches this element. Lippert and Budenske each disclose a method of operating a heterogeneous computing system to compute a computation task. Lippert discloses operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, with at least one of the nodes being arranged to compute a computation task comprising a plurality of sub-tasks. Specifically, Lippert discloses a "computer cluster arrangement for processing a computation task," where the "computer cluster arrangement" comprises a plurality of computation nodes and a plurality of boosters (booster nodes). EX1004, [0009]-[0013]. Processors applied in computation nodes are different from the processors applied in boosters. EX1004, [0029]. A POSITA thus would have understood that the computer cluster arrangement is a heterogeneous computing system because the computer cluster

[2] Reference numbers in the format of [claim#.element#] are added throughout for ease of reference.

arrangement comprises different types of processors, arranged as computation nodes and booster nodes. Figure 2 of Lippert, reproduced below, "shows a computer cluster arrangement comprising a cluster C as well as a booster group BG," where the cluster as depicted comprises "four computation nodes, also referred to as CN, as well as three boosters, also referred to as B." EX1004, [0067]. While Figure 2 labels the computation nodes as the cluster "C," a POSITA would understand that the entire arrangement depicted in Figure 2 is the cluster, as stated in Lippert: "FIG. 2 shows a schematic illustration of a computer cluster arrangement." EX1004, [0059].

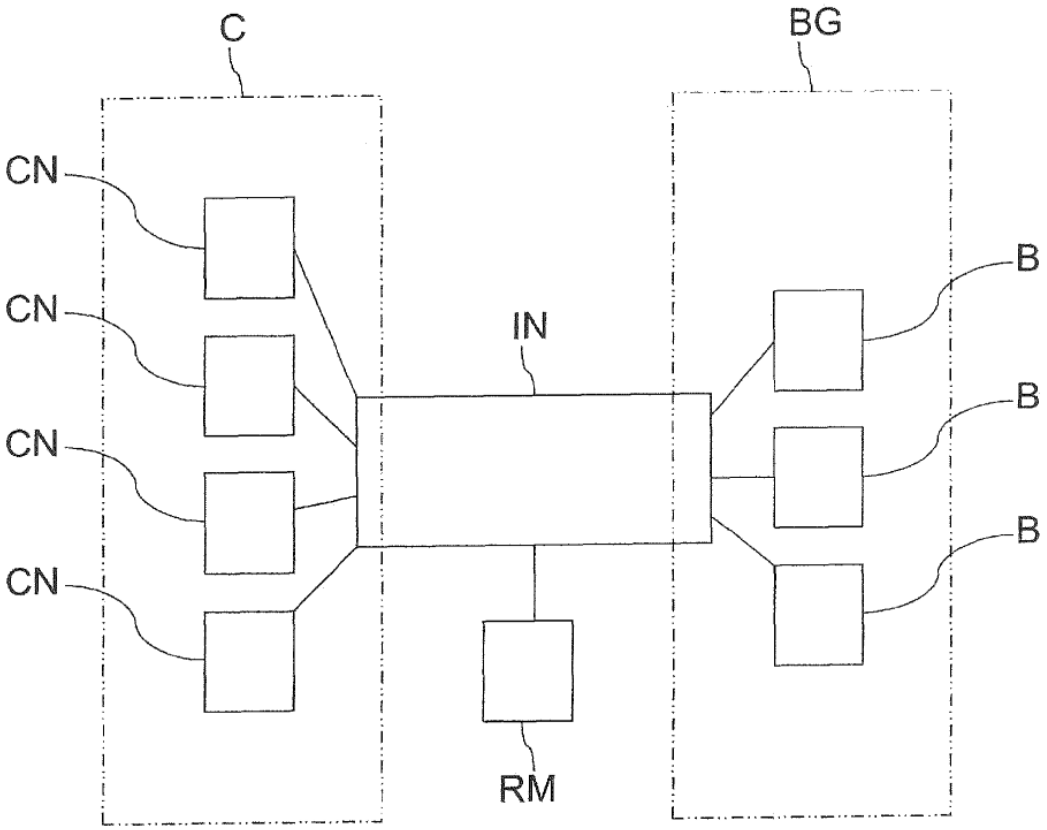


Fig.: 2

EX1004, FIG. 2.

69. Lippert discloses that "the computation task may comprise several sub[-]problems, also referred to as sub[-]tasks, which in their entirety describe the overall computation task." EX1004, [0016]. The computation task may be divided "into several parts," and the computer cluster arrangement is arranged "to solve the parts of the computation task in parallel or in succession." EX1004, [0016]. Thus, Lippert (and Lippert-Budenske) discloses this claim element.

70. Budenske discloses a method for operating a heterogeneous computing system to compute a computation task, the computation task comprising a plurality of sub-tasks. Budenske discloses a "methodology for real-time on-line input-data dependent remapping of the application subtasks to the processors in the heterogeneous parallel hardware platform." EX1005, 387. Budenske explains that the configuration of the heterogeneous parallel hardware platform (heterogeneous computing system) depends on the expected needs of the application (task) to be computed, and as an example, teaches a heterogeneous parallel hardware platform that includes "up to four different types of processors, and up to a total of 64 processors (of all types combined)." EX1005, 391.

71. As explained by Budenske, an "application … can be modeled as an iterative execution of a set of partially ordered subtasks." EX1005, 390. An "application task" is represented as "a *DDG* (data dependency graph), whose nodes

are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies among subtasks." EX1005, 391. The DDG is "structured as a directed acyclic graph (*DAG*)." EX1005, 391.

72. In combination, Lippert-Budeske discloses a method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks.

b) Element [1.2]

[1.2] in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and

73. Lippert-Budenske discloses and teaches this element. Lippert discloses computing a "first part of the computation task" and a "second part of the computation task" (*the plurality of sub-tasks*) by, respectively, "at least two of the plurality of computation nodes CN" (*at least a portion of the plurality of computation nodes*) and "at least one booster B" (*at least a portion of the plurality of booster nodes*). EX1004, [0091]. A POSITA would have understood this computing of the parts of the computation task by at least two of the plurality of computation nodes and at least one booster as *assigning and processing* the parts of the computation task *in a first distribution*, at least because Lippert teaches

"establishing a static assignment [of a booster to a computation node] at start of a processing of a computation task," where "assignment information [is provided] to the computation nodes for outsourcing parts of the computation tasks from at least one computation node to at least one booster." EX1004, [0013]-[0014]. A POSITA would have understood "distribution" as used here in the claim to refer to how the sub-tasks are assigned to nodes (and thus distributed among the nodes), as opposed to meaning a probability distribution like a Gaussian.

74. Although Lippert discloses the computation node "outsourcing" parts of the computation task to a booster where the computation node processes the first part in a first step and the booster node processes the second part in a second step (e.g., EX1004, [0091]-[0092]), suggesting that the parts are solved in succession, Lippert also discloses that the system may "solve the parts of the computation task *in parallel* or in succession." EX1004, [0016] (emphasis added). Further, Lippert does not expressly specifically disclose the iterative execution of a computation task, and thus does not expressly specifically disclose a "first computing iteration," but Lippert explains that "the person skilled in the art appreciates that any of the steps can be performed iteratively, in a different order and may comprise further sub steps." EX1004, [0093].

75. Budenske discloses assigning a plurality of sub-tasks to processing resources of a heterogeneous computing system for processing in a first iteration.

For example, Budenske discloses, "[f]or the initial iteration through the set of subtasks" (*in a first computing iteration*), "assign[ing] resources (e.g., processors) to the subtasks." EX1005, 391; *see also id.*, 396-97 (describing how to assign sub-tasks to processors). This "initial mapping" (*first distribution*) of the subtasks (*the plurality of sub-tasks*) to the processing resources is used for a first computing iteration of the application (*computation task*). EX1005, 395.

76. A POSITA would have implemented Lippert-Budenske to, in a first computing iteration of an iterative application, assign the plurality of sub-tasks to at least a portion of the computation nodes and at least a portion of the boosters according to an initial mapping, where the initial mapping maps sub-tasks to processors (e.g., computation nodes or boosters), and where their assigned computation nodes and boosters thus process the plurality of sub-tasks. The POSITA would have been motivated to implement Lippert-Budenske in this way at least because doing so would ensure that sub-tasks are assigned to processing resources "best suited" to compute the algorithms of the sub-tasks while accounting for the initial architecture of computation nodes and boosters, and characteristics of the application itself (e.g., what algorithms will be used to compute each sub-task; what are the data dependencies between sub-tasks; etc.). *E.g.*, EX1005, 392, 396-397. Lippert-Budenske thus teaches and renders obvious, in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the

plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution.

c) Element [1.3]

[1.3] generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

77. Lippert-Budenske discloses and teaches this element. After processing the sub-tasks with the "initial mapping," Budenske teaches "monitor[ing] the run-time values of the dynamic parameters at the end of each iteration through the underlying DDG to make a heuristically-based decision whether to continue with the current mapping or to select and instantiate a new mapping (for the next iteration)." EX1005, 395. Budenske thus discloses using values of dynamic parameters (*information relating to the processing of the sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes*) to generate a "new assignment" (*further distribution*) of sub-tasks to resources "to use for the next execution iteration through the subtasks" (*in a further computing iteration*). EX1005, 391.

78. Specifically, "[a]fter each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image

stream being processed." EX1005, 391. The dynamic parameters are "computed by the application as it executes." EX1005, 393. The values for the dynamic parameters are thus obtained after processing the sub-tasks in an iteration: "After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel." EX1005, 391. At each iteration, there can be a different mapping of the DDG (parallelism) onto the accelerators. The latest values of dynamic parameters are an example of the claim's "information relating to the processing of the plurality of sub-tasks." The "most recent values of such dynamic parameters" are then used to determine whether "to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration," and "[i]f it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks." EX1005, 391. In particular, Budenske teaches generating a "vector of *D* representative values" as an "approximation of the set of current actual dynamic parameters," and using that vector to identify the optimal mapping of sub-tasks from the possible mappings (tracked in a "Mapped DDG" or "MDDG Table"). EX1005, 402. Budenske thus teaches selecting a new mapping from a set of candidate mappings that best matches the values of the dynamic parameters (*the information*), and instantiating (*generating*) the further distribution according to the selected mapping for the next computing iteration.

EX1005, 402-403. Budenske also discloses an alternative wherein a fully "on-line heuristic" is used, such that the new mapping is generated in real-time between iterations based on the information. EX1005, 389-90, 398, 403.

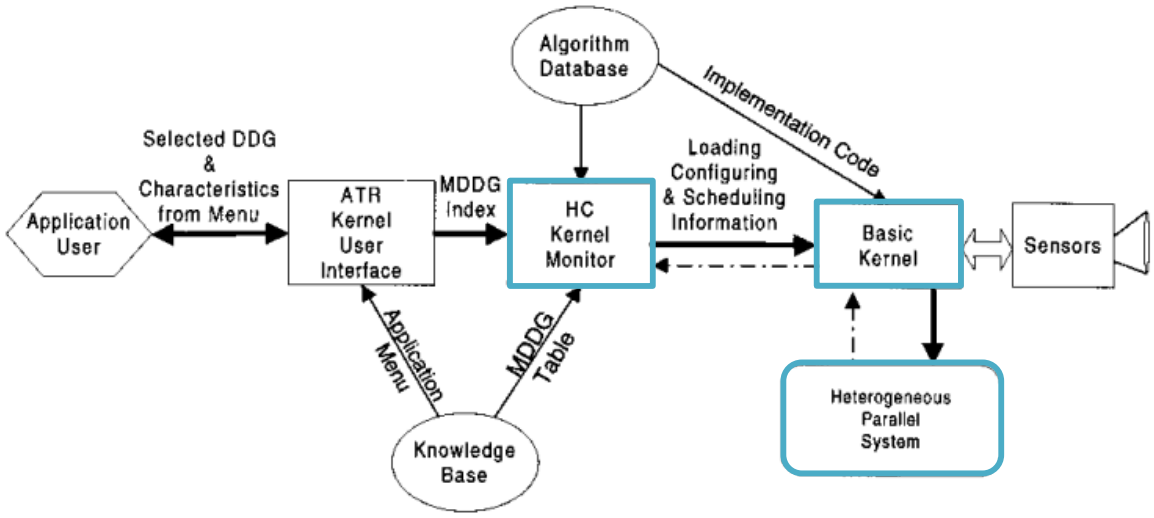
79. A POSITA would have found it obvious to implement the heterogeneous computing system of Lippert with the dynamic sub-task mapping method of Budenske such that, between iterations, Lippert-Budenske uses values of dynamic parameters calculated during a preceding iteration to generate a new assignment of sub-tasks to the plurality of computation nodes and the plurality of boosters for processing in the next iteration. The POSITA would have been motivated to implement Lippert-Budenske in this way partly in order "to reduce execution time of the next iteration." EX1005, 391. The POSITA would have further been motivated to implement Lippert-Budenske in this way because Budenske explains that, "[b]ecause the execution time of application subtasks … is highly input-data dependent …, this matching and scheduling of application subtasks to processors *must* be performed dynamically at run time." EX1005, 390 (emphasis added). The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske partly because Lippert and Budenske each provide ample implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. Lippert-Budenske thus teaches and renders this element and this claimed method obvious.

2. Claim 2

2. The method according to claim 1, wherein an application manager receives the information and determines the further distribution.

80. Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske renders claim 1 obvious, as discussed above in Section X.C.1. Budenske discloses an "HC Kernel" (also referred to as "HC Kernel Monitor") and a "Basic Kernel" that cooperate to manage an application, and thus a POSITA would have understood aspects of the HC Kernel and the Basic Kernel together as an *application manager*. EX1005, 388-89, 401-02. According to Budenske, "the HC Kernel interacts with the Basic Kernel … to execute the application and monitor its execution so it can decide when to remap the subtasks onto the hardware platform" (EX1005, 388), where the Basic Kernel receives "the latest values of … dynamic parameters" at the end of each iteration (*receives the information*). EX1005, 402 (describing how the Basic Kernel "begin[s] execution of the application task"). The Basic Kernel sends the latest values of the dynamic parameters to an HC Kernel Monitor, which then "use[s] the most recent values of these dynamic parameters to estimate if changing the matching and scheduling will reduce the expected execution time of the next iteration through the corresponding DDG." EX1005, 402. Figure 3 of Budenske depicts the Basic Kernel and the HC Kernel Monitor (together, *the application*

manager) interacting with the heterogeneous parallel computing system during execution of an iterative application:



EX1005, Figure 3 (annotated); *see also id.*, Figure 1.

81. A POSITA would have understood the HC Kernel and the Basic Kernel of Budenske together as an "application manager" that "receives the information and determines the further distribution" because the Basic Kernel receives the values of the dynamic parameters (*receives the information*) from the heterogeneous parallel computing system and inputs the dynamic parameter values to the HC Kernel to determine a new assignment of sub-tasks to processing resources (*determines the further distribution*). EX1005, 391, 402. Lippert-Budenske thus discloses and teaches this claim element and renders this claim obvious.

3. Claim 3

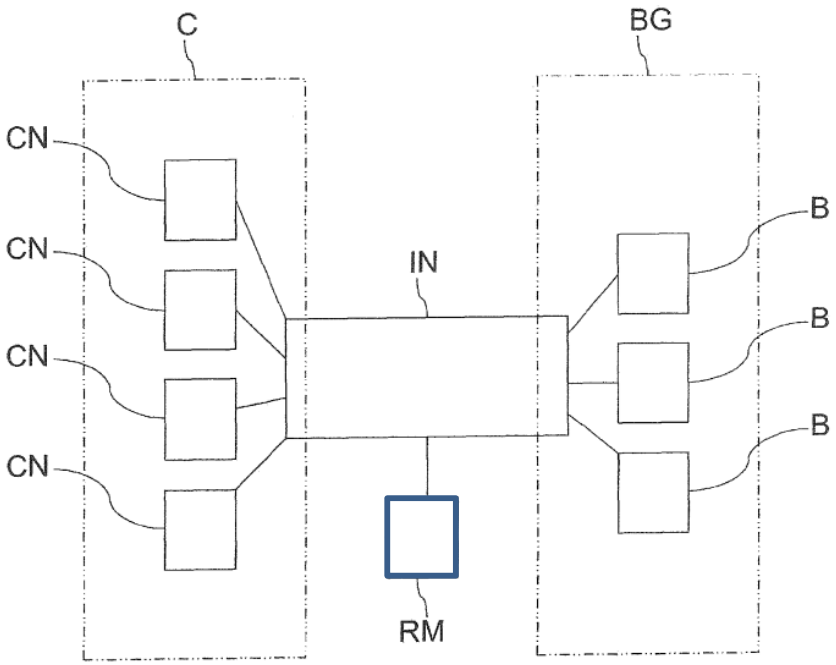
a) Element [3.1]

[3.1] The method according to claim 2, wherein a resource manager determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task and

82. Lippert-Budenske discloses and teaches this claim element and renders this claim obvious for the foregoing and the following reasons. Specifically, as discussed below, Lippert discloses a "resource manager," and Budenske discloses an HC Kernel Monitor, and a POSITA implementing Lippert-Budenske would combine aspects of Lippert's resource manager and Budenske's HC Kernel Monitor into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task. Lippert-Budenske thus includes a resource manager that manages resources of the heterogeneous computing system (e.g., computation nodes, booster nodes, and the assignment thereof for processing a computation task), and an application manager that manages applications (e.g., computation tasks) processed by the heterogeneous computation system, where the resource manager handles an initial mapping of sub-tasks because it manages the initial arrangement of the resources and does not depend on information from processing the sub-tasks. In contrast, the application manager

handles re-mapping of the sub-tasks to the resources because it receives information relating to the processing of the sub-tasks during an application's execution. *See* ¶¶80-81 (discussing the application manager of Lippert-Budenske); ¶¶83-85 (discussing the resource manager of Lippert-Budenske).

83. Lippert discloses a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes as a function of the computation task. The resource manager "establish[es] a static assignment at start of a processing of a computation task," and "establish[es] a dynamic assignment at runtime, which means during processing of the computation task." EX1004, [0013]; *id.*, [0014], [0017]. FIG. 2 of Lippert depicts the resource manager RM of the heterogeneous computing system.



EX1004, FIG. 2 (annotated). This resource manager RM is similar to the resource manager RM of the '442 patent. *See* EX1001, FIG. 1 (element 28). The *resource manager* of Lippert determines the computation node-booster assignment based on "computation task requirements," and thus *as a function of the computation task*. EX1004, [0017]. Specifically, "booster allocation is performed *as a function of application needs*, which means in dependency of processing a specific computation task." EX1004, [0015] (emphasis added). The resource manager "determin[es] an initial assignment of boosters to computation nodes" based on a predetermined assignment metric, and dynamically assigns boosters to computation nodes "at runtime" as the computation task requirements evolve and the assignment metric is accordingly altered. EX1004, [0017]; *id.*, [0013], [0030]. The assignment metric is managed by the resource manager. It is updated "as a function of a load balancing, which detects workload of the computer cluster arrangement, especially of the boosters." *Id.*, [0017]. For the assignment metric, Lippert teaches "detect[ing] computing capacities of boosters and further detect[ing] computation task requirements," so that a booster assigned to a computation node "provides the required capacities to the computation node." *Id.* A POSITA would have understood the resource manager's assignment based on "computation task requirements" as being determined *as a function of the computation task*. EX1004, [0017].

84. Budenske, meanwhile, teaches an HC Kernel Monitor that determines the assignment of the plurality of sub-tasks to the plurality of nodes for the first computing iteration as a function of the computation task. Specifically, Budenske teaches an HC Kernel Monitor that "establish[es] the initial mapping" of sub-tasks to processing resources for an initial iteration (*the assignment of the plurality of sub-tasks to the plurality of computation nodes … for the first computing iteration*). EX1005, 402. The initial mapping is a mapped DDG (MDDG) for a given initial scenario (i.e., set of dynamic parameter values), where the MDDG is structured as a directed acyclic graph that arranges sub-tasks according to "data dependencies among subtasks" with algorithms selected for implementing the sub-tasks (*as a function of the computation task*). *Id.*, 391, 396, 402. A POSITA would have understood the HC Kernel Monitor partly as a *resource manager* because it "decide[s] how to assign resources (e.g., processors) to the subtasks." EX1005, 391.

85. In combination, Lippert-Budenske includes a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes, as taught by Lippert, for the first computing iteration, as taught by Budenske. The assignment is determined as a function of the computation task, as taught by both Lippert and Budenske, because doing so would "enforce data-dependency constraints among the subtasks" (EX1005, 397) and ensure that the sub-tasks of an iterative application are assigned

"to the processors where [they are] best suited for execution" (EX1005, 390). *See also* EX1004, [0015], [0017].

b) Element [3.2]

[3.2] wherein the application manager receives the information and processes the information as input to the resource manager such that the resource manager dynamically alters further distribution during the computing of the computation task.

86. Lippert-Budenske discloses and teaches this element, thus rendering the claimed method obvious for the foregoing and the following reasons. As discussed above in Section X.C.2, Lippert-Budenske includes an application manager that receives the information (*the application manager receives the information*) and determines the further distribution (*processes the information*). Lippert-Budenske further includes a resource manager that "establish[es] a dynamic assignment at runtime, which means during processing of the computation task" (*dynamically alters further distribution during the computing of the computation task*), as taught by Lippert. EX1004, [0013]; Section X.C.3.a).

87. A POSITA implementing Lippert-Budenske would have found it obvious to input the new mapping of sub-tasks by the application manager into the resource manager such that the resource manager dynamically alters the distribution of computation nodes and booster nodes during the computing of the computation task, because doing so would utilize the "maximum flexibility in providing hardware

resources" taught by Lippert. EX1004, [0015]; *see also* EX1005, 390 (teaching to "fully utilize the architectural flexibility"). That is, in addition to re-mapping the sub-tasks to processing elements of the system as taught by Budenske, Lippert-Budenske also dynamically assigns computation nodes and boosters to each other "during processing of the computation task" as taught by Lippert (EX1004, [0013]). By providing the new mapping for the next iteration to the resource manager, Lippert-Budenske would, for example, allocate additional boosters for processing a given sub-task or otherwise re-configure computation node-booster assignment to implement the new mapping and "reduce execution time of the next iteration" (EX1005, 391). EX1004, [0014]-[0017], [0068]. In this way, Lippert-Budenske "provide[s] fault tolerance in case of accelerator failures" and avoids "a lack of resources or … an excessive supply of resources" when executing an application. EX1004, [0003].

88. A POSITA would have understood that "the resource manager dynamically alters further distribution" as recited in this claim element *encompasses* "the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other," because claim 4 depending from claim 3 specifies this, and further because the '442 patent describes the resource manager adjusting booster assignments according to updated assignment

information from the application manager. EX1001, 4:27-31. Thus, in view of the above, Lippert-Budenske discloses and teaches this element and this claim.

4. Claim 4

> 4. The method according to claim 3, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

89. Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and the following reasons. As discussed above in Section X.C.3, the resource manager of Lippert-Budenske receives the information and dynamically alters the assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. EX1004, [0013], [0017].

5. Claim 5

> 5. The method according to claim 1, wherein daemons operate in the plurality of computation nodes and the plurality of booster nodes to generate the information.

90. Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons addressing claim 1 and for the following reasons. Lippert-Budenske renders claim 1 obvious. Section X.C.1. Lippert discloses "detect[ing] computing capacities of boosters" (EX1004, [0017]), and Budenske teaches values of dynamic parameters being reported to the Basic Kernel

during each iteration (*e.g.*, EX1005, 402). In order to send the dynamic parameter values taught by Budenske, as well as information regarding computing capacities as taught by Lippert, from each computation node and each booster node to a central manager to determine a re-mapping of sub-tasks, a POSITA thus would have implemented Lippert-Budenske with daemons at each computation node and each booster node configured to collect and send this information. POSITAs knew that a daemon is a background process that performs a specific task, and that daemons were commonly implemented at compute nodes to execute tasks at the compute nodes and report status information back to a central resource manager for managing cluster resources. *E.g.*, EX1013, 5-7 (describing a central controller daemon and daemons running at each compute node "to export control to SLURM"), 20 (describing a daemon at each node "for managing user jobs and monitoring system state," where its "most common action … is to report system state on request"); EX1021, 7:66-67 (defining a "daemon" as "a program that runs unattended to perform a standard service"), 8:2-3 ("Daemon processes generally provide services that must be available at all times."), 2:54-3:18 (describing using daemons to facilitate communication in a "parallel processing environment" comprising "a plurality of nodes connected through a network to one another"). A POSITA thus would have been motivated to implement background processes (i.e., daemons) operating at each node that communicate the relevant information in a status update

for dynamically remapping the sub-tasks among the nodes, to facilitate inter-element communication while providing communication authentication beneficially, and to enable independent control of task execution and reporting of task processing at each node. *E.g.*, EX1012, 9 (describing a central daemon and node daemons or "TaskTrackers" for executing tasks at the node), 38-40 (describing a "NodeManager" daemon at each node "responsible for launching the applications' containers, monitoring their resource usage …, and reporting the same to the ResourceManager"); EX1013, 10 (describing how daemons authenticate and secure communications between nodes); EX1022, 3:63-66 ("As is commonly known, a daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events."), 3:66-4:14 (explaining that "[t]ypical daemon processes … perform administrative tasks" and are implemented in "any environment … which is capable of implementing background processes and supports communications between the various Nodes of the Cluster(s)").

6. Claim 6

6. The method according to claim 1, wherein the first distribution is determined based on a rating provided in source code for each sub-task in the plurality of sub-tasks.

91. Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the above reasons associated with claim 1 and the following reasons. Lippert-Budenske renders claim 1 obvious. Section X.C.1. Lippert

discloses determining the first distribution based on a rating provided in the source code for each sub-task in the plurality of sub-tasks.

92. Specifically, regarding *a rating*, Lippert discloses priority, complexity, and scalability information (individually and together, "*a rating*") for each sub-task because the sub-tasks comprise the priority information, complexity information, and scalability information. Specifically, Lippert discloses that "[s]pecific *computation tasks may comprise priority information*, which indicates how urgently this specific computation task has to be computed." EX1004, [0020] (emphasis added). This priority indicates "how urgent a processing of a computation task, or at least a part of a computation task, is compared to other parts of computation tasks being originated from other computation nodes." *Id.* This relates to the dependency between sub-tasks, where the processing of one sub-task depends on the processing of another sub-task. Regarding complexity information and scalability information, Lippert discloses applications or computation tasks where "[e]ach code has highly scalable and less scalable complex elements," broken down into "exascale code blocks" and "complex code blocks," respectively (i.e., scalable sub-tasks and less-scalable complex sub-tasks). EX1004, [0084]. As "the distinction between highly scalable and complex is made on the level of code blocks," the code blocks (sub-tasks) include the complexity information and scalability information. *Id.* Each sub-

task of the computation task thus comprises priority information, complexity information, and scalability information (*a rating*).

93. The rating is *provided in the source code for each sub-task in the plurality of sub-tasks* because the computation task "comprises" the information and is "defined by means of … a source code." EX1004, [0016].

94. Regarding *the first distribution is determined based on a rating*, Lippert further discloses a "predetermined assignment metric" for assigning nodes to each other, with parts of the computation task allocated accordingly, being specified as a function of "complexity information, scalability information," and "priority information" (*a rating*). EX1004, [0023]; *id.*, [0018], [0043]. This information is specific to each sub-task, and the assignment in Lippert, including the initial distribution of sub-tasks among the nodes, is based partly on this information. *E.g.*, *id.*, [0015] ("booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"), [0017]-[0020] (describing using the assignment metric to assign nodes). Lippert teaches adapting the architecture of the computation nodes and the boosters of a computing cluster to account for the scalability of the code blocks. EX1004, [0084]-[0088].

95. A POSITA would have understood that Lippert thus discloses and teaches determining the first distribution of sub-tasks based on a rating provided in source code for each sub-task, including a rating based on scalability information

(e.g., indicating that the code block/sub-task is more scalable or less scalable) to ensure that the part of a computation task distributed to a booster is suitable for the resource capacities of the booster. This is consistent with the disclosure in the '442 patent of an operator estimating a scalability factor for each sub-task and distributing the sub-tasks accordingly. EX1001, 4:16-47. Further, the '442 patent confirms that POSITAs already knew how to annotate application code to indicate that a given task is "highly scalable" and thus suitable for execution by boosters. EX1001, 1:39-55 (citing EX1014); EX1014, 15-16 (describing how to designate portions of code as sub-tasks as highly scalable and suitable for execution on specified hardware devices, e.g., boosters).

96. A POSITA would have found it obvious when implementing Lippert-Budenske to determine the initial mapping of sub-tasks to computation nodes and boosters based on a rating provided in source code for each sub-task, where such rating comprises an indication of scalability, as taught by Lippert. The POSITA would have been motivated to do so to create optimal assignments of sub-tasks to processing resources (e.g., computation nodes or boosters) that account for the suitability of the processing resource for the corresponding scalability/complexity of the sub-task while minimizing the expected execution time. The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske in this way because Lippert provides ample implementation details for achieving the disclosed

functionality, and a POSITA would have found it simple and straightforward to combine the teachings. Lippert-Budenske thus renders this claim obvious.

7. Claim 7

7. The method according to claim 1, wherein the information is used to provide a grouping of sub-tasks in at least one of the first computing iteration and the further computing iteration.

97. Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons associated with claim 1 and the following reasons. Lippert-Budenske renders claim 1 obvious. Section X.C.1. Regarding *a grouping of sub-tasks*, Lippert discloses "divid[ing] the computation task into several parts," including a first part computed by computation node(s) and a second part computed by booster(s). EX1004, [0016]; *id.*, [0009]-[0012]. A POSITA would have understood these "parts" of the computation task as groupings of sub-tasks, e.g., a first grouping of sub-tasks for processing by computation nodes and a second grouping of sub-tasks for processing by boosters, because Lippert discloses that a computation task "comprise[s] several sub problems, also referred to as sub tasks." EX1004, [0016].

98. As discussed above in Section X.C.1.c), Lippert-Budenske discloses and teaches using information relating to processing the plurality of sub-tasks to generate a further distribution of the sub-tasks for a further computing iteration. Given this disclosure and teaching, a POSITA implementing Lippert-Budenske

would have been motivated, with a reasonable expectation of success, to use the values of the dynamic parameters (*the information*) from a preceding iteration to provide, via the new mapping of sub-tasks, a first grouping of sub-tasks and a second grouping of sub-tasks for processing respectively with computation node(s) and booster(s) in the next iteration (*the further computing iteration*), at least because Lippert expressly discloses providing such groupings of sub-tasks. EX1004, [0016]; *id.*, [0009]-[0012].

8. Claim 8

8. The method according to claim 3, wherein a daemon operating at a node generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks.

99. Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the reasons associated with claims 1-3 and the following reasons. Lippert-Budenske renders claim 3 obvious, and further renders the daemons operating at the plurality of computation nodes and booster nodes obvious. Sections X.C.3, X.C.5. Lippert discloses "detect[ing] workload of the computer cluster arrangement, especially of the boosters," as well as "computing capacities." EX1004, [0017]. For Lippert's resource manager to detect such information, a POSITA would have understood that a background process (i.e., a daemon) implemented at each node to measure and report workload and computing capacity information (*generates a measure of a loading of the node during the processing of*

a sub-task of the plurality of sub-tasks) would be the most common way to generate and communicate such information, and the easiest to implement because daemons already exist for performing this function. *E.g.*, EX1012, 38 (describing a per-node daemon "monitoring their resource usage (CPU, memory, disk, network) and reporting the same to the ResourceManager"); EX1013, 6-7 (describing a "daemon running on each compute node"), 19-20 (explaining how node daemons provide "resource consumption information … associated with a job").

100. A POSITA thus would have been motivated, with a reasonable expectation of success, to implement Lippert-Budenske with a daemon operating at each computation node and booster node that generates a measure of the loading (workload) of nodes during processing of a sub-task of the plurality of sub-tasks, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system, as taught by Budenske, to also account for "load balancing" as taught by Lippert. EX1004, [0017].

9. Claim 9

[9.1] A heterogeneous computing system comprising:

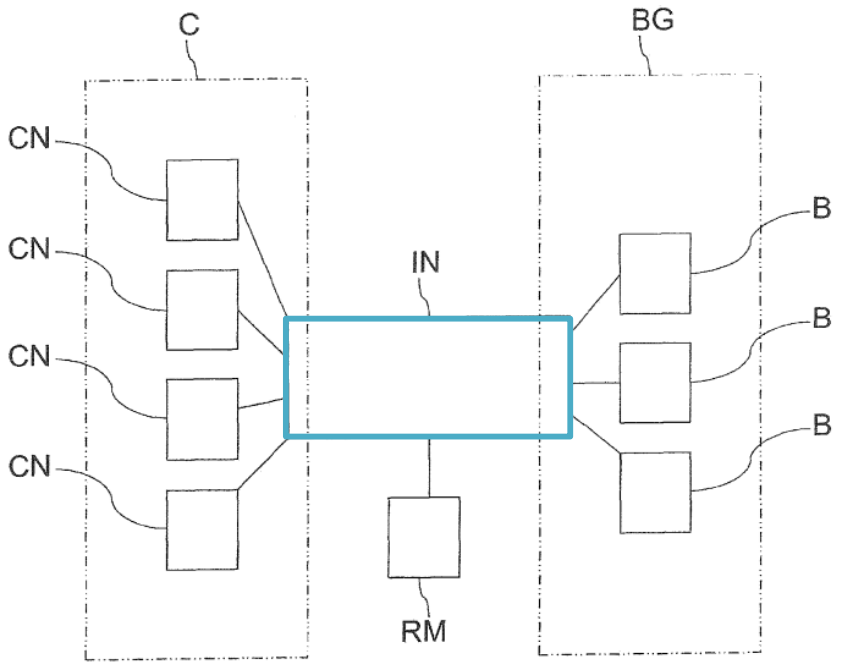
101. Lippert-Budenske discloses and teaches this element because Lippert and Budenske disclose a heterogeneous computing system. Section X.C.1.a).

[9.2] a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks;

102. Lippert-Budenske discloses and teaches this element because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. Section X.C.1.a).

[9.3] a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes;

103. Lippert-Budenske discloses and teaches this element because Lippert discloses the plurality of computation nodes and the plurality of booster nodes "interfacing a communication infrastructure." EX1004, [0010]-[0011]; *id.*, [0016]. Thus, "the computation nodes as well as the boosters interact by means of the communication infrastructure." *Id.*, [0016]. FIG. 1 of Lippert depicts the communication infrastructure IN, also called an interconnect, interfacing with each computation node CN and each booster B. *Id.*, [0067], FIG. 1.



EX1004, FIG. 1 (annotated).

104. Budenske similarly discloses that "[a]ll the processors of all types will have communication paths to one another," via a "bus [that] can be used to provide communications among different collections of processors." EX1005, 392.

105. Thus, when implementing Lippert-Budenske, a POSITA would have found it obvious for the heterogeneous computing system to include a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes, at least because Lippert and Budenske each disclose such a communication interface.

[9.4] a resource manager for assigning at least a portion of the plurality of booster nodes and at least a portion of the plurality of computation nodes to each other for the computing of the one or more tasks in a first computing iteration; and

106. Lippert-Budenske discloses and teaches this element. Sections X.C.3, X.C.4.

[9.5] an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

107. Lippert-Budenske discloses and teaches this element. Section X.C.2. Lippert-Budenske thus discloses and teaches all elements and renders this claimed system obvious.

10. Claim 10

10. The heterogeneous computing system according to claim 9, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other.

108. Lippert-Budenske discloses and teaches this element, and thus renders this claim obvious. Sections X.C.9, X.C.4.

XI. CLAIMS 2-5 AND 8-10 ARE OBVIOUS IN VIEW OF LIPPERT, BUDENSKE, AND KAMBATLA

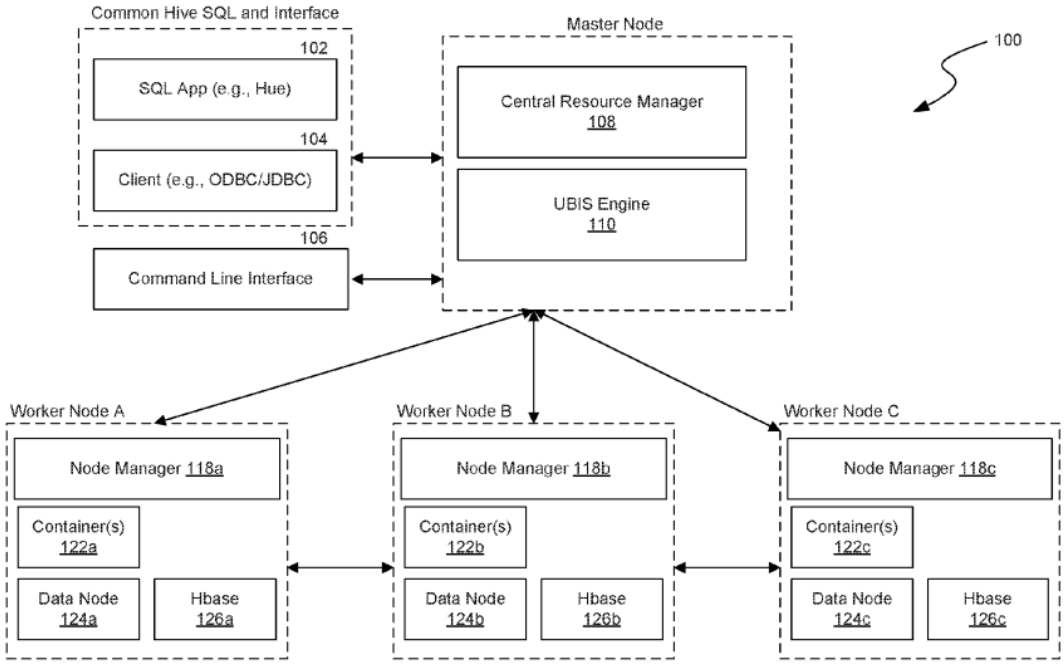
109. As explained below, claims 2-5 and 8-10 are obvious in view of Lippert, Budenske, and Kambatla. While Lippert and Budenske, in combination, render claims 1-10 obvious, the express teachings of Kambatla regarding resource management daemons reflect the knowledge of a POSITA and further confirm the obviousness of the claims.

A. Kambatla (EX1006)

110. Kambatla is a U.S. patent application filed May 15, 2017, and claiming priority to a provisional application filed September 14, 2016, and I understand this means that Kambatla qualifies as prior art under at least Section 102(a)(2). EX1006, Cover.

111. Kambatla discloses a resource management system for the "allocation and management of computing resources in distributed computing clusters." EX1006, [0002]. Kambatla teaches techniques referred to as "utilization-based incremental scheduling" (UBIS) that "opportunistically allocate[s] computing resources not utilized by prior allocations." EX1006, [0017]. Kambatla proposes UBIS as an improvement to known "cluster scheduler" frameworks such as Apache Hadoop YARN ("Yet Another Resource Negotiator"), used for scheduling (i.e., assigning) multiple tasks of a job to cluster computing resources. EX1006, [0004], [0020] ("UBIS can be implemented in or for use with Apache YARN"), [0023]-[0024], [0085].

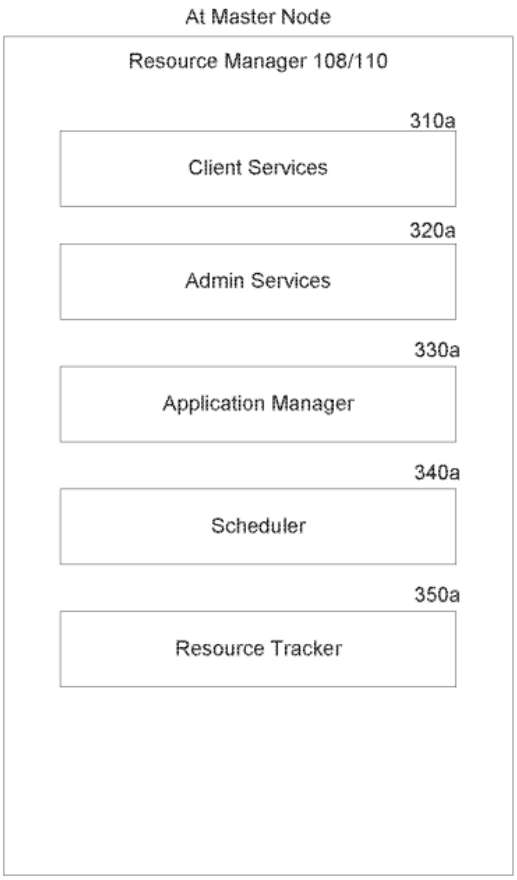
112. Kambatla discloses a computing environment, depicted in Figure 1, comprising a master node and a plurality of worker nodes in communication, where the master node assigns tasks of a job to the computing resources of the worker nodes for execution. EX1006, [0021]-[0022].



EX1006, FIG. 1. To facilitate allocating computing resources for tasks, Kambatla discloses daemons at each computing node, including a central resource manager daemon at the master node and a node manager daemon at each worker node. EX1006, [0030]-[0036] (describing resource manager), [0037]-[0041] (describing node managers), FIGS. 3A-B. The resource manager is "configured to manage and arbitrate resources among applications in the system." EX1006, [0023].

113. Although Figure 1 depicts the resource manager 108 and the UBIS engine 110 as separate components of the master node, Figure 3A of Kambatla discloses a single resource manager comprising both components. EX1006, [0023], [0030], FIG. 3A. The resource manager comprises, among other things, "an

application manager 330a, a scheduler 340a, and a resource tracker 350a." EX1006, [0031].



EX1006, FIG. 3A.

B. Combining Lippert, Budenske, And Kambatla

114. A POSITA would have found it obvious to implement a combination of Lippert and Budenske, as discussed in Section X.C. A POSITA further would have been motivated, with a reasonable expectation of success, to combine the teachings of Lippert-Budenske with Kambatla's resource scheduling daemons. A skilled artisan would have combined these respective teachings into an improved

design (hereinafter "Lippert-Budenske-Kambatla") that included benefits from each reference. The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
  - A plurality of computation nodes;
  - A plurality of booster nodes; and
  - A communication infrastructure interfacing each of the computation nodes and the booster nodes;
- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401); and
  - Dynamically assign, during the processing of the computation task, computation nodes and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);
- An application manager module, as taught by Kambatla, arranged to:

- Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);
- Monitor execution of sub-tasks to collect information relating to the processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
- Assign, according to a new mapping obtained using information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-402);

- A node manager daemon at each computation node and booster node, as taught by Kambatla, arranged to:
  - Generate information relating to the processing of a sub-task at the node; and
  - Report the information to the application manager module.

115. A POSITA would have been motivated, with a reasonable expectation of success, to combine Lippert, Budenske, and Kambatla by implementing the computer cluster of Lippert with the dynamic re-mapping of sub-tasks between

iterations of an iterative application as taught by Budenske, and with the resource management daemons as taught by Kambatla, for at least the following reasons.

116. *First*, Kambatla, like Lippert-Budenske, discloses a heterogeneous computing system that processes a task comprising a plurality of sub-tasks. EX1006, [0004], [0021], [0059]. Kambatla further teaches that such systems enable the processing of not just one task or application but multiple tasks concurrently, with the system's processing resources being shared by the multiple tasks. It also provides techniques for optimally scheduling tasks across the processing resources. EX1006, [0004], [0017]. A POSITA thus would have been motivated to incorporate the complementary teachings of Kambatla into Lippert-Budenske to leverage better the full processing capacity of the heterogeneous computing system.

117. *Second*, a POSITA implementing Lippert-Budenske would have recognized the need to facilitate inter-element communication to allow the resource manager to allocate sub-tasks to nodes efficiently, and Kambatla teaches how to do so. For example, while Lippert discloses the computation nodes, booster nodes, and resource manager communicating with each other via the communication infrastructure (*e.g.*, EX1004, [0016]), Kambatla describes a node manager daemon operating at each node to communicate with a central resource manager daemon and to manage the execution of tasks at the node. EX1006, [0023], [0038]-[0041]. A POSITA thus would have naturally turned to the teachings of Kambatla for

implementation details on communicating information (i.e., values of dynamic parameters, processing capacities of the node) to the central resource manager.

118. *Third*, the particular teachings of Kambatla follow the same principles as Lippert-Budenske regarding the utilization of processing resources. For example, Kambatla describes the prior art background of allocating cluster computing resources to "jobs [that] typically include multiple tasks," and the problem of "resource fragmentation and severe under-utilization of the computing resources in the cluster." EX1006, [0003]-[0005]; *see also* EX1004, [0005] (discussing the problem of "over- or under subscription of accelerators Acc depending on the computation task"). This occurs because "[t]he amount of computing resources required to process a given task can be difficult to predict," as "[i]t is inevitably difficult to accurately estimate the resource requirements of a job or its constituent tasks because: (i) resource usage of a task varies over time, and (ii) resource usage can vary across tasks of the same job based on the input they process." EX1006, [0005]. Kambatla addresses this problem by monitoring "actual utilization of resources at a computing node" via a YARN-based resource manager that communicates with node managers at nodes of a cluster, and "improv[ing] effective resource utilization in a distributed computing cluster." EX1006, [0017], [0042]; *id.*, [0018]-[0023], [0030]-[0036] (describing a central resource manager daemon), [0037]-[0041] (describing a node daemon), [0042]-[0049] (describing "Optimizing

Resource Utilization"). This utilization-based resource allocation for applications deployed to a computer cluster is complementary to the dynamic assignments of Lippert-Budenske. *E.g.*, EX1004, [0015] (teaching "booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"); EX1005, 390-391 (describing "input-data dependent" execution of sub-tasks and the remapping of sub-tasks to processors as dynamic parameters evolve over time). Thus, a POSITA seeking to fully leverage the processing capacity of a computer cluster, for example by processing multiple iterative applications concurrently, would naturally seek out and incorporate teachings on cluster resource management daemons that facilitate dynamic resource allocation, such as those taught by Kambatla.

119. *Finally*, a POSITA had a reasonable expectation of success implementing Lippert-Budenske-Kambatla, at least because a POSITA was familiar with and understood how to implement "known resource management techniques (such as in YARN)" (EX1006, [0023]), and because doing so would involve writing only minor software code that was well within the skill of a POSITA. *See, e.g.*, EX1012, xviii (explaining that "YARN makes the effort of building [distributed computing] systems significantly simpler"). Further, a POSITA would have reasonably expected to succeed in incorporating Kambatla's teachings into Lippert-Budenske because Kambatla explains that its teachings can be successfully

implemented in heterogeneous computing systems comprising "CPU and memory resources" (i.e., computation nodes) and "GPU[s] (graphical processing [units])" (i.e., boosters). EX1006, [0020]; *id.*, [0023], [0059]. Further, the '442 patent admits that using daemons at nodes to facilitate communication was already known, confirming that a POSITA knew how to implement such features. EX1001, 3:6-18, 3:39-43, 4:32-37.

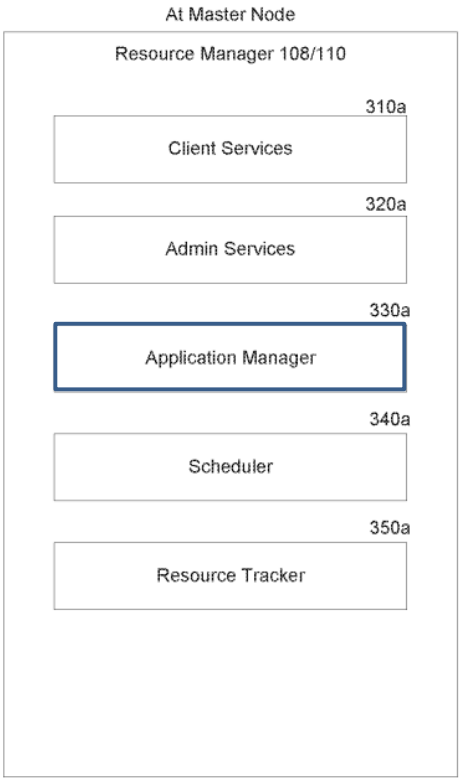
120. Thus, a POSITA implementing the dynamic assignment of sub-tasks between iterations, as taught by Budenske, to computation nodes and boosters of a heterogeneous computing system, as taught by Lippert, would have been motivated to implement a cluster scheduling framework as taught by Kambatla, in order "to improve effective resource utilization" and thus minimize "processing duration." EX1006, [0042].

1. Claim 2

2. The method according to claim 1, wherein an application manager receives the information and determines the further distribution.

121. Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and the following reasons. Lippert-Budenske renders claim 1 obvious, as discussed above in Section X.C.1. Further, Lippert-Budenske discloses and teaches this element. Section X.C.2.

122. Moreover, Kambatla expressly discloses various modules of a central resource manager that a POSITA would have understood as the "application manager" recited in the claim. Kambatla discloses an "application manager 330a … responsible for maintaining a collection of submitted 'applications' or 'jobs.'" EX1006, [0034]. The application manager is a module of a "resource manager daemon." EX1006, [0031]. The application manager includes an "application master service responsible for communicating with all the application masters at the" computing nodes. EX1006, [0034]. FIG. 3A of Kambatla depicts the resource manager 108 including the application manager 330a.



EX1006, FIG. 3A (annotated). Further, the resource tracker 350a of the resource manager "monitors available resources at the nodes, by receiving status updates from the worker nodes." EX1006, [0036]; *id.*, [0058] (describing how the resource manager "can actively monitor resource usage of each container at the worker nodes and of the worker nodes themselves," via "heartbeat information" that includes "utilization information"). At the same time, "[t]he scheduler 340a is responsible for allocating resources to the various applications." EX1006, [0035]. Despite being depicted as separate components, Kambatla teaches that the resource manager can "include more or fewer components, organized in various configurations." EX1006, [0030]. A POSITA thus would have understood that the functions of the application manager 330a and the resource tracker 350a were combinable into a single component (*application manager*) that manages an application deployed to cluster resources and communicates with the resources, as well as provides information from the resources to the scheduler 340a of the resource manager to allocate sub-tasks of the application to the resources for processing.

123. In combination, a POSITA implementing Lippert-Budenske-Kambatla would have found it obvious to use an application manager as taught by Kambatla to perform the functions of the Basic Kernel of Budenske, at least because the application manager and the Basic Kernel similarly manage the execution of an application on a plurality of computing devices. Further, a POSITA would have been

motivated to implement the dynamic assignment of sub-tasks to the compute nodes and booster nodes of a heterogeneous computing system, as taught by Lippert-Budenske, with the application manager of Kambatla to efficiently enable and manage the execution of multiple such applications on the plurality of computing resources. *E.g.*, EX1006, [0004], [0023]-[0024], [0034].

124. The POSITA would have reasonably expected to succeed because implementing such an application manager would entail only providing simple software code and configurations to perform the known functions, which was within the skill of a POSITA. *E.g.*, EX1006, [0101] (explaining broadly how "the routines executed to implement the embodiments of the disclosure[] may be implemented"); EX1012, 59-84 (detailing how to install and configure YARN on a cluster).

2. Claim 3

a) Element 3.1

3. The method according to claim 2, wherein a resource manager determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task and

125. Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Section XI.B.1. Lippert-Budenske discloses and teaches this element, as discussed above in Section X.C.3. Further, Kambatla discloses a "central resource manager [that] is a general

resource manager configured to manage and arbitrate resources among applications in the system." EX1006, [0023], FIGS. 1, 3A. A POSITA implementing Lippert-Budenske-Kambatla would have combined the resource manager of Lippert-Budenske with the central resource manager of Kambatla into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task, as taught by Lippert-Budenske. Section X.C.3.a). The POSITA would have done so to gain the additional advantages of Kambatla, namely to enable efficient scheduling of multiple tasks, rather than one computational task, across the processing resources of the heterogeneous computing system, while minimizing "processing duration." EX1006, [0042], [0004], [0017], [0023]-[0024].

b) <u>Element 3.2</u>

wherein the application manager receives the information and processes the information as input to the resource manager such that the resource manager dynamically alters further distribution during the computing of the computation task.

126. Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and the following reasons. Lippert-Budenske-Kambatla includes an *application manager* that receives information from the nodes and processes the information to update an assignment of sub-tasks

to the nodes. Section XI.B.1. Lippert-Budenske-Kambatla further includes a *resource manager*. Section XI.B.2.a). Lippert-Budenske includes a resource manager that dynamically alters further distribution of computation nodes and booster nodes during the computing of the computation task. Section X.C.3.b). Further, Kambatla discloses that its central resource manager includes various modules configured to handle different tasks and feed information to other modules "to aid in resource allocation." EX1006, [0036], [0031]-[0035]. Kambatla also discloses that "more or fewer components, organized in various configurations … remain[] within the scope" of Kambatla. EX1006, [0030]. A POSITA thus would have naturally implemented Lippert-Budenske-Kambatla with a central resource manager as taught by Kambatla that includes a module configured to perform the dynamic-assignment functions of Lippert-Budenske's resource manager, thus achieving the benefits of Kambatla's resource manager and the dynamic assignment of Lippert-Budenske's resource manager. Sections X.C.3.b), XI.B. So implemented, the resource manager of Lippert-Budenske-Kambatla uses the information input by the application manager to dynamically alter further distribution during the computing of the computation task, thus combining the functions of the resource manager of Lippert, the HC Kernel Monitor and Basic Kernel of Budenske, and the resource manager of Kambatla.

3. Claim 4

4. The method according to claim 3, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

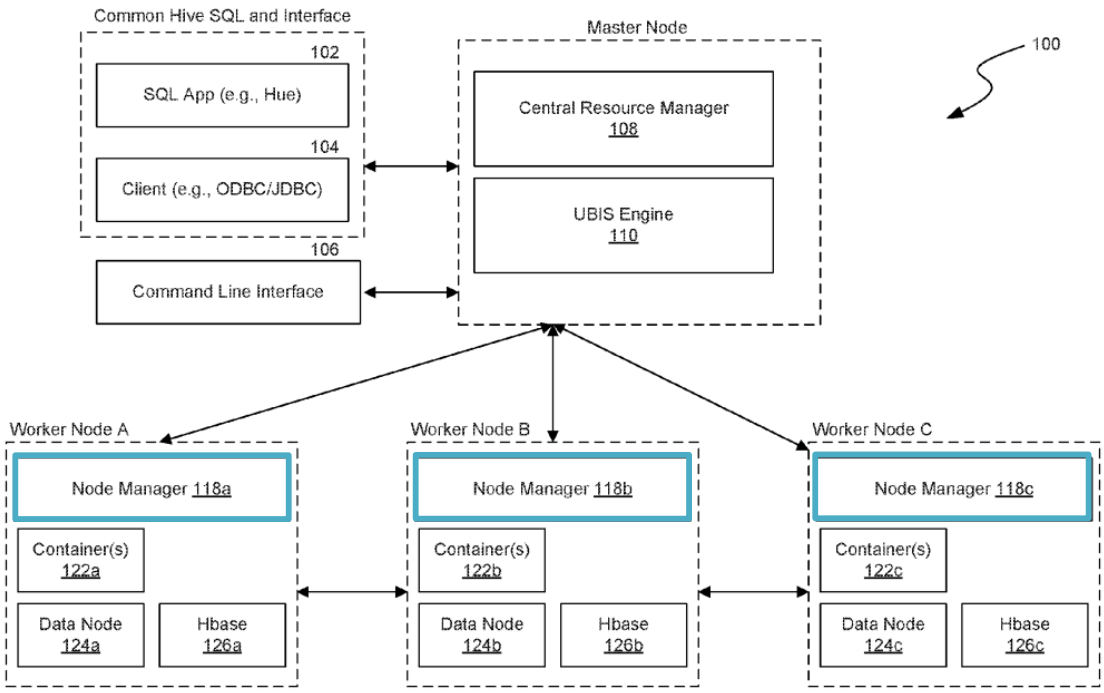
127. Lippert-Budenske-Kambatla teaches and discloses this element, and renders this claim obvious, at least because Lippert-Budenske teaches and discloses a resource manager that receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. Sections XI.B.2, X.C.4. As Lippert-Budenske-Kambatla includes a resource manager that combines the central resource manager of Kambatla with the resource manager of Lippert-Budenske, the resource manager of Lippert-Budenske-Kambatla receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

4. Claim 5

5. The method according to claim 1, wherein daemons operate in the plurality of computation nodes and the plurality of booster nodes to generate the information.

128. Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. Lippert-Budenske renders claim 1 obvious. Section X.C.1. Further, Lippert-Budenske discloses and teaches this element. Section X.C.5.

Kambatla provides further implementation details by expressly disclosing a "resource manager daemon installed at worker nodes in a distributed computing cluster." EX1006, [0037]. The "resource manager daemon installed at the worker nodes includes a node manager 310b, and one or more application masters 320b." EX1006, [0038], FIGS. 1, 3B. The central resource manager "[c]ommunicat[es] with node managers 118a-c which act as the agents at each node," to "allocate and schedule resources available at the various nodes based on the available resources reported from each node manager." EX1006, [0023].



EX1006, FIG. 1 (annotated). Each node manager "registers with the [central] resource manager and broadcasts the [node's status] including the status of available resources … at the node." EX1006, [0039]. Further, "[a]pplication masters 320b are

responsible for requesting resources from the resource manager 108/110, working with node managers 310b to manage and monitor the containers 330b allocated by the resource manager 108/110." EX1006, [0041]. Containers are an abstraction of the worker's nodes computing resources allocated to process a task. EX1006, [0004], [0023].

129. In combination, Lippert-Budenske-Kambatla includes daemons as taught by Kambatla at each worker node—i.e., at each computation node and each booster node—to generate dynamic parameter information as taught by Budenske, as well as information regarding computing capacities and computation task requirements as taught by Lippert. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way because doing so would facilitate efficient communication of the relevant information in a status update for dynamically remapping the sub-tasks among the nodes. In contrast, the "additional information [taught by Budenske and Lippert] will typically add little overhead to an existing node [status update]." EX1006, [0039].

5. Claim 8

8. The method according to claim 3, wherein a daemon operating at a node generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks.

130. Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. Section XI.B.2. Lippert-Budenske teaches and discloses

this element. Section X.C.8. Further, according to Kambatla, the daemon operating at a node "monitors the containers 330b at the node … for resource utilization while processes are running" and "calculate[s] the aggregate resource utilization across all containers in the worker node" (*generates a measure of a loading of the node during processing*). EX1006, [0038]-[0040].

131. In combination, Lippert-Budenske-Kambatla includes a daemon operating at each computation node and booster node that generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system as taught by Budenske to also account for "load balancing" as taught by Lippert. EX1004, [0017].

6. Claim 9

[9.1] A heterogeneous computing system comprising:

132. Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert and Budenske each disclose a heterogeneous computing system. Section X.C.1.a).

[9.2] a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks;

133. Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. Section X.C.1.a).

[9.3] a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes;

134. Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert-Budenske discloses and teaches this element. Section X.C.9. Kambatla also discloses a "cluster of worker nodes in [] communication (e.g., via a computer network) with each other and one or more master nodes." EX1006, [0021].

135. In combination, Lippert-Budenske-Kambatla includes a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes.

[9.4] a resource manager for assigning at least a portion of the plurality of booster nodes and at least a portion of the plurality of computation nodes to each other for the computing of the one or more tasks in a first computing iteration; and

136. Lippert-Budenske-Kambatla discloses and teaches this element. Sections X.C.1, XI.B.1, XI.B.2.

[9.5] an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

137. Lippert-Budenske-Kambatla discloses and teaches this element. Sections X.C.1, XI.B.1, XI.B.4.

7. Claim 10

10. The heterogeneous computing system according to claim 9, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other.

138. Lippert-Budenske-Kambatla discloses and teaches this claim. Sections XI.B.6, XI.B.3.

I, Darrell Long, do hereby declare and state, that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code.

Dated: December 20, 2024

Darrell D. E. Long, Ph.D.

APPENDIX A

Cumulative Biobibliography
University of California, Santa Cruz

April 27, 2024

DARRELL DON EARL LONG

Distinguished Research Professor of Engineering
Kumar Malavalli Endowed Chair Professor in Storage Systems Research, Emeritus

Employment History

2024–	Distinguished Visiting Scholar, Santa Clara University
2023–	Distinguished Research Professor of Engineering, University of California, Santa Cruz
	Distinguished Professor of Engineering, *Emeritus*, University of California, Santa Cruz
2019–23	Director, Center for Research Systems & Storage, University of California, Santa Cruz
2017–23	Distinguished Professor of Engineering, University of California, Santa Cruz
2005–23	Kumar Malavalli Endowed Chair Professor in Storage Systems Research
2004–10	Associate Dean for Research and Graduate Studies, Jack Baskin School of Engineering, University of California, Santa Cruz
2001–19	Director, Storage Systems Research Center, University of California, Santa Cruz
1999–17	Professor, Computer Science, University of California, Santa Cruz
1998–01	Associate Dean, Jack Baskin School of Engineering, University of California, Santa Cruz
1994–99	Associate Professor, Computer Science, University of California, Santa Cruz
1988–94	Assistant Professor, Computer Science, University of California, Santa Cruz
1986–88	Research Assistant, Computer Science & Engineering, University of California, San Diego
1985–87	Teaching Associate, Computer Science & Engineering, University of California, San Diego
1984–87	Lecturer in Mathematics, Department of Mathematical Sciences, San Diego State University
1981–84	Systems Programmer, University Computer Center, San Diego State University

Visitor History

2019 (Winter)	Professeur Invité, Sorbonne Université
2016–	Associate Member, European Organization for Nuclear Research (CERN)
2016 (Fall)	Professeur Invité, Conservatoire National des Arts et Métiers
2014–2020	Visiting Scientist, Lawrence Livermore National Laboratory
2014 (Fall)	Professeur Invité, Conservatoire National des Arts et Métiers
	Professeur Invité, Université Paris–Descartes

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 92 of 411

2013 (Winter)	Professeur Invité, Conservatoire National des Arts et Métiers
	Professeur Invité, Université Paris–Descartes
	Professeur Invité, Université Paris–Dauphine
2012–2014	Visiting Professor, United States Naval Postgraduate School
2011 (Winter)	Professeur Invité, Conservatoire National des Arts et Métiers
2010 (Winter)	Professeur Invité, Université Paris–Dauphine
2009 (Winter)	Professeur Invité, Université Paris–Dauphine
2008 (Winter)	Visiting Professor, University of Technology, Sydney
2007 (Winter)	Visiting Scholar, University of California, San Diego
	Visiting Scholar, Center for Communications Research
1995–2011	Visiting Scientist, IBM Almaden Research Center

Education

Ph.D.	1988	University of California, San Diego, Computer Science
M.S.	1986	University of California, San Diego, Computer Science
B.S.	1984	San Diego State University, Computer Science

Honors and Awards

2018	IBM Faculty Award
2016	Best Paper Award, "RESAR: Reliable Storage at Exabyte Scale," *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*
2015	Best Paper Award, "Classifying Data to Reduce Long Term Data Movement in Shingled Write Disks," *Conference on Mass Storage Systems and Technologies*
2013	Best Short Paper Award, "A File By Any Other Name: Managing File Names with Metadata," *International Systems and Storage Conference*
2012	Certificate of Appreciation for Outstanding Service, National Research Council
	IEEE Certificate of Appreciation
2011	Chancellor's Achievement Award for Diversity
2010	Professor *ad Honorem* de la Universidad Católica del Uruguay
2008	Fellow, American Association for the Advancement of Science
	Certificate of Appreciation for Outstanding Service, National Research Council
2006	Fellow, Institute of Electrical and Electronics Engineers
2005	Kumar Malavalli Endowed Chair in Storage Systems Research
	IBM Research Invention Achievement Award (Third Plateau)
	Certificate of Appreciation for Outstanding Service, National Research Council

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 93 of 411

2003 IBM Faculty Award

2002 IBM Research Invention Achievement Award (Second Plateau)

IEEE Computer Society Certificate of Appreciation

2001 IBM Corporate Accomplishment Award for Adaptive Differential Back-up in the Tivoli Storage Manager

1997 IBM Research Invention Achievement Award (First Plateau)

1996 IBM Research Invention Achievement Award

IEEE Computer Society Certificate of Appreciation

1995 Honorable mention, 1994–95 Excellence in Teaching Award

Best Paper Award, "A Longitudinal Study of Internet Host Reliability," *Symposium on Reliable Distributed Systems*

1994 Senior Member, Institute of Electrical and Electronics Engineers

1993 IEEE Computer Society Certificate of Appreciation

1992 Regents Junior Faculty Fellow

1991 Student Alumni Council Favorite Professor Award

1989 Regents Junior Faculty Fellow

1988 Regents Junior Faculty Fellow

Grants

2022–25 *Trust Worthy Information Storage Technology Enhanced Devices (TWISTED)*, Defense Advanced Research Projects Agency (DARPA), Phase 2: $2,500,000 (with P. Alvaro). **Panel Reviewed**.

2020–21 *Trust Worthy Information Storage Technology Enhanced Devices (TWISTED)*, Defense Advanced Research Projects Agency (DARPA), Phase 1: $225,000. **Panel Reviewed**.

2019–24 *Phase II I/UCRC CRSS*: *Center for Research in Storage Systems*, National Science Foundation, $500,000. **Peer Reviewed**.

2018–21 *A Multi-Layered Deniable Steganographic File System*, National Science Foundation, $500,000. **Peer Reviewed**.

2017–20 *Integrated End-to-end Performance Prediction and Diagnosis for Extreme Scientific Workflows*, Department of Energy, Office of Science, $300,000. **Panel Reviewed**.

2015–17 *Automatic Tuning and Contention Management for Lustre*, Intel Corporation, $187,024. **Panel Reviewed**.

2015–18 *Automatic Storage and Network Contention Management for Large-scale High-Performance Computing Systems*, National Science Foundation, $450,000. **Peer Reviewed**.

2013–18 *I/UCRC: A Single-Site I/UCRC Center for Research in Storage Systems*, National Science Foundation, $468,850 (with E. Miller). **Peer Reviewed**.

2012–13 *Better File System Management Through Rich Metadata and Provenance*, NASA, $40,000. **Panel Reviewed**.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 94 of 411

2012–15 RESAR: *Robust, Efficient, Scalable, Autonomous Reliable Storage for the Cloud*, National Science Foundation, $374,700. **Peer Reviewed**.

2011–12 *Reducing Fragmentation in Deduplicated Storage Systems*, Hewlett-Packard Laboratories, $50,000. **Panel Reviewed**.

Better File System Management Through Rich Metadata and Provenance, NASA, $40,000. **Panel Reviewed**.

Improving Automated Metadata Extraction, Analysis and Reporting, United States Navy, $150,000. **Panel Reviewed**.

2010–11 *Metadata Exploration in Digital Forensics*, United States Navy, $196,156. **Panel Reviewed**.

2010–13 *Dynamic Non-Hierarchical File Systems for Exascale Storage*, Department of Energy, Office of Science, $1,462,106. **Peer Reviewed**.

2010–13 *LockBox*: *Enabling Users to Keep Data Safe*, National Science Foundation, $499,907 (with E. Miller). **Peer Reviewed**.

2009–14 *Collaborative Research*: *A Multi-University I/UCRC Center on Intelligent Storage*, National Science Foundation, $275,000 (with E. Miller). **Peer Reviewed**.

2009–12 *Scalable Data Management Using Metadata and Provenance*, National Science Foundation, $553,000 (with E. Miller, in collaboration with Margo Seltzer, Harvard University). **Peer Reviewed**.

2010–11 *Trading Storage for Computation,* NASA, $61,000. **Panel Reviewed**.

2009–10 *Development of a Collaborative Project for Remotely Sensed Science and Technology,* NASA, $300,000 (with Raphael Kudela, Ethan Miller, Roberto Manduci, Donald Potts, Eli Silver, Michael Loik and Chris Wilmers). **Panel Reviewed**.

2007 *ViewFS: Dynamic Name-Spaces for Metadata-Rich File Systems,* Lawrence Livermore National Laboratory, $74,994 (with E. Miller). **Panel Reviewed**.

2006–09 *End-to-End Performance Management for Large Distributed Storage,* National Science Foundation, $956,647 (with S. Brandt). **Peer Reviewed**.

2006–11 *Petascale Data Storage Institute,* Department of Energy, Office of Science, $11,250,000 (in collaboration with Carnegie-Mellon University, University of Michigan, and the Department of Energy National Laboratories). **Peer Reviewed**.

2005–10 *Institute for Scalable Scientific Data Management,* Department of Energy, Los Alamos National Laboratory, $4,750,000 (with S. Brandt and E. Miller). **Panel Reviewed**.

2005–06 *Scalable File Systems for High Performance Computing,* Department of Energy, Defense Programs Laboratories, $250,000 (with S. Brandt, E. Miller, M. Abadi and C. Maltzahn). **Peer Reviewed**.

2003 *A Scalable On-Line Associative Deep Store,* University of California, Microelectronics Electronics Innovation and Computer Research Opportunities Program (MICRO), $27,000. **Panel Reviewed**.

2003–06 *A Scalable On-Line Associative Deep Store,* National Science Foundation, $294,012. **Peer Reviewed**.

2002–07 *Applications of Data Grouping for Effective Mobility,* National Science Foundation, $338,130. **Peer Reviewed**.

2002–05 *Scalable File Systems for High Performance Computing,* Department of Energy, Defense Programs Laboratories, $900,000 (with S. Brandt, E. Miller and K. Obraczka). **Panel Reviewed**.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 95 of 411

2001–02 *Building a High-Performance Storage Systems from Commodity Components,* Department of Energy, Lawrence Livermore National Laboratory, $65,000. **Panel Reviewed**.

2000–01 *Application of Probe-based Storage to High Performance Computing,* Department of Energy, Lawrence Livermore National Laboratory, $39,239. **Panel Reviewed**.

2000–03 *Architectures and Algorithms to Exploit Probe-Based Storage,* National Science Foundation, $345,191 (with S. Brandt and T. Madhyastha). **Peer Reviewed**.

2000–02 *An Experimental Study of Broadcasting Protocols for Video-on-Demand,* National Science Foundation, $100,000. **Peer Reviewed**.

1999–01 *High Performance Integration of Advanced Tertiary Stores,* National Science Foundation, $170,353. **Peer Reviewed**.

1998 *Real-Time Environmental Information Network and Analysis System* (REINAS), Office of Naval Research, $100,000. **Panel Reviewed**.

1998–99 *Tactical Environmental Data System/Real-time* (TEDS/RT), Naval Research Laboratory, $75,000. **Panel Reviewed**

1997–02 *National Partnership for Advanced Computing Infrastructure* (NPACI), National Science Foundation, $819,000 (with P. Mantey, A. Pang and S. Flatté). **Panel Reviewed**.

1997–99 *Predictive Power Conservation,* National Science Foundation, $122,011 (with D. Helmbold). **Peer Reviewed**.

1997–99 *Improving Cache Performance by Predicting I/O System Actions,* National Science Foundation, $136,647. **Peer Reviewed**.

1996–97 *Efficient Back-up and Restore Using Differential Files,* International Business Machines Corporation, $79,864.

1995–98 *Instant Infrastructure and Distributed Resource Management,* Office of Naval Research, $90,911. **Panel Reviewed**.

1995–96 *Efficient Resource Utilization Through Idle-Time Prediction,* University of California Seed Funds, $15,000 (with D. Helmbold).

1993–94 *Global-scale Distributed Data Base Management,* Academic Senate Committee on Research, $1,500.

1993–95 *Integration of Heterogeneous Real-time Data Repositories for Scientific Use,* Office of Naval Research, $219,164. **Panel Reviewed**.

1992–93 *Estimating Host Reliability Using the Internet,* University of California, Microelectronics Electronics Innovation and Computer Research Opportunities Program (MICRO), $5,481. **Panel Reviewed**.

1992 University of California, Regents Junior Faculty Fellowship, $700.

1992–97 *Real-time Environmental Information Network and Analysis System* (REINAS), Office of Naval Research, $4,767,112 (with P. Mantey *et al.*). **Peer Reviewed**.

1992–93 University of California, Regents Junior Faculty Fellowship, $750.

1992–93 *Estimating Reliability Using the Internet,* University of California Seed Funds, $10,000.

1992–93 *Estimating Host Reliability Using the Internet,* Sun Microsystems, Incorporated, $16,590. **Panel Reviewed**.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 96 of 411

1991–92 *A Distributed Architecture for High-speed I/O,* Department of Energy, Lawrence Livermore National Laboratory, $15,183. **Panel Reviewed**.

1991–93 *High-speed Distributed Storage Management,* National Science Foundation, $68,955. **Peer Reviewed**.

1991–92 *A High-speed System to Support Multimedia,* Academic Senate Committee on Research, $1,500.

1991–92 *Instructional Improvement Grant,* Academic Senate Committee on Teaching, $200.

1990–91 *Concurrent Systems Laboratory,* University of California Seed Funds, $12,500.

1989–90 *Fault-tolerant Operating System Techniques,* University of California, Microelectronics Electronics Innovation and Computer Research Opportunities Program, (MICRO) $4,000. **Panel Reviewed**.

1989 University of California, Regents Junior Faculty Fellowship, $500.

1988–89 *Modeling Fault-tolerant Computations,* Academic Senate Committee on Research, $1,500.

1988 University of California, Regents Junior Faculty Fellowship, $1,000.

Scholarly and Creative Work

Books

1. Ruth A. David, Steven R.J. Brueck, Stephen W. Drew, Alan H. Epstein, Robert A. Furman, Sharon Glotzer, Christopher C. Green, Diane E. Griffin, J. Jerome Holton, Michael R. Ladisch, Darrell Long, Frederick R. Lopez, Richard M. Osgood, Stewart D. Personick, Alton D. Romig, S. Shankar Sasrty, James B. Smith, Camillo J. Taylor and Diane Wiley. *Avoiding Surprise in an Era of Global Technology Advances.* National Research Council, National Academies Press, 2005. **Peer Reviewed**.

2. John L. Carroll and Darrell D. E. Long. *Theory of Finite Automata.* Englewood Cliffs, New Jersey: Prentice-Hall, 1989. **Editorial Review**.

Chapters in Books

1. Ismail Ari, Ahmed Amer, Ethan L. Miller, Robert Gramacy, Scott A. Brandt and Darrell D. E. Long. "ACME: Adaptive Caching using Multiple Experts," *Distributed Data & Structures 4* (Proceedings in Informatics 14), W. Litwin and G. Lèvy, Eds. Paris, France: Carleton Scientific, March 2002, pp. 143–158. **Peer Reviewed**.

2. Steven W. Carter, Darrell D. E. Long and Jehan-François Pâris. "Video-on-Demand Broadcasting Protocols," *Multimedia Communications: Directions and Innovations,* J. D. Gibson, Ed. San Diego: Academic Press, 2000, pp. 179–189. **Peer Reviewed**.

3. Richard A. Golding and Darrell D. E. Long. "Simulation Modeling of Weak-Consistency Protocols," *Network Systems Design,* E. Gelenbe, K. Bagchi and G. Zobrist, Eds. Gordon and Breach Science Publishers, 1999, pp. 161–186. **Invited**, **Peer Reviewed**.

4. Darrell D. E. Long and John L. Carroll. "Using Simulation to Evaluate Consistency Protocols for Replicated Data," *Progress in Simulation II,* G. W. Zobrist and J. V. Leonard, Eds. Ablex Publishing, 1995. **Invited**, **Peer Reviewed**.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 97 of 411

Peer Reviewed Journals

1. Liangkang Zhang, Yulai Xie, Minpeng Jin, Pan Zhou, Gongming Xu, Yafeng Wu, Dan Feng, Dan and Darrell Long. "A Novel Hybrid Model for Docker Container Workload Prediction," *IEEE Transactions on Network and Service Management, to appear.*

2. Yafeng Wu, Yulai Xie, Xuelong Liao, Pan Zhou, Dan Feng, Lin Wu, Xuan Li, Wildani, Avani and Darrell Long. "Paradise: Real-time, Generalized, and Distributed Provenance-Based Intrusion Detection," *ACM Transactions on Dependable and Secure Computing, to appear.*

3. Daniel Bittman, Peter Alvaro, Pankaj Mehra, Darrell D. E. Long and Ethan L. Miller. "Twizzler: A Data-centric OS for Non-volatile Memory," *ACM Transactions on Storage*, vol. 17, no. 2, June 2021.

4. Austen Barker, Yash Gupta, James Hughes, Ethan L. Miller and Darrell D. E. Long. "Rethinking the Adversary and Operational Characteristics of Deniable Storage," *Journal Journal of Surveillance, Security and Safety,* vol. 2, May 27, 2021.

5. Yulai Xie, Minpeng Jin, Zhuping Zou, Gongming Xu, Dan Feng, Wenmao Liu and Darrell Long. "Real-time Prediction of Docker Container Resource Load Based on A Hybrid Model of ARIMA and Triple Exponential Smoothing," *IEEE Transactions on Cloud Computing*, April 22, 2020.

6. Yulai Xie, Yafeng Wu, Dan Feng and Darrell Long. "P-Gaussian: Provenance-Based Gaussian Distribution for Detecting Intrusion Behavior Variants Using High Efficient and Real Time Memory Databases," *IEEE Transactions on Dependable and Secure Computing*, December 17, 2019.

7. Yulai Xie, Zhuping Zhou, Kai Huang, Gongming Xu, Dan Feng and Darrell Long. "A Docker Container Anomaly Monitoring System Based on Optimized Isolation Forest," *IEEE Transactions on Cloud Computing*, September, 2018.

8. Die Hu, Dan Feng, Yulai Xie, Gongming Xu, Xinrui Gu and Darrell Long. "Efficient Provenance Management via Clustering and Hybrid Storage in Big Data Environments," *IEEE Transactions on Big Data*, December 2020, vo. 6, no. 4, pp. 792–803.

9. Yulai Xie, Dan Feng, Yuchong Hu, Yan Li, Staunton Sample and Darrell Long. "Pagoda: A Hybrid Approach to Enable Efficient Real-time Provenance Based Intrusion Detection in Big Data Environments," *IEEE Transactions on Dependable and Secure Computing*, August 29, 2018.

10. Stephanie N. Jones, Ahmed Amer, Ethan L. Miller, Darrell D. E. Long, Rekha Pitchumani and Christina R. Strong. "Classifying Data to Reduce Long-Term Data Movement in Shingled Write Disks," *ACM Transactions on Storage*, vol. 12 no. 1, February 2016.

11. Yulai Xie, Dan Feng, Yan Li and Darrell D. E. Long. "Oasis: an Active Storage Framework for Object Storage Platform," *Future Generation Computer Systems*, vol. 56, March 2016, pp. 746–758.

12. Nicolás López, Matías Rodríguez, Catalina Fellegi, Darrell Long and Thomas Schwarz, S.J. "Even or Odd: A Simple Graphical Authentication System (*Par o Impar*: *un sistema gráfico simple de autenticación*)," *IEEE Latin America Transactions* (Revista IEEE América Latina), 13(3), pp. 804–809, March 2015.

13. Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Yan Li and Darrell D. E. Long. "Evaluation of a Hybrid Approach for Efficient Provenance Storage," *ACM Transactions on Storage*, vol. 9 no. 4, November 2013.

14. Witold Litwin, Darrell Long and Thomas Schwarz. "Combining Chunk Boundary and Chunk Signature Calculations for Deduplication (*Combinar la calculacion de limites de trozos y de firmas de trozos para deduplicacion*)," *IEEE Latin America Transactions* (Revista IEEE América Latina), 10(1), pp. 1305–1311, January 2012.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 98 of 411

15. Ahmed Amer, JoAnne Holliday, Darrell D. E. Long, Ethan L. Miller, Jehan-François Pâris and Thomas Schwarz, S.J. "Data Management and Layout for Shingled Magnetic Recording," *IEEE Transactions on Magnetics*, vol. 47, no. 10, pp. 3691–3697, October 2011.

16. Lawrence L. You, Kristal T. Pollack, Darrell D. E. Long and K. Gopinath. "PRESIDIO: A Framework for Efficient Archival Data Storage," *ACM Transactions on Storage*, vol. 7, no. 2, art. 6, pp. 1–60, July 2011.

17. Bo Hong, Scott A. Brandt, Darrell D. E. Long, Ethan L. Miller and Ying Lin. "Using MEMS-Based Storage in Computer Systems – Device Modeling and Management," *ACM Transactions on Storage*, vol. 2, no. 2, May 2006, pp. 139–160.

18. Bo Hong, Feng Wang, Scott A. Brandt, Darrell D. E. Long and Thomas J. E. Schwarz, S.J. "Using MEMS-Based Storage in Computer Systems – MEMS Storage Architectures," *ACM Transactions on Storage*, vol. 2, no. 1, February 2006, pp. 1–21.

19. Randal Burns, Larry Stockmeyer and Darrell Long. "In-Place Reconstruction of Version Differences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, July 2003, pp. 973–984.

20. Miklós Ajtai, Randal Burns, Ronald Fagin, Darrell Long and Larry Stockmeyer. "Compactly Encoding Unstructured Inputs with Differential Compression," *Journal of the ACM*, vol. 49, no. 3, May 2002, pp. 318–367.

21. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "Efficient Data Distribution in a Web Server Farm," *IEEE Internet Computing*, vol. 5, no. 4, July 2001, pp. 56–65.

22. Randal C. Burns, Robert M. Rees, Larry Stockmeyer, and Darrell D. E. Long. "Scalable Session Locking for a Distributed File System," *Cluster Computing Journal*, Kluwer Academic Publishers, vol. 4, no. 4, October 2001, pp. 295–306.

23. David P. Helmbold, Darrell D. E. Long, Tracey L. Sconyers and Bruce Sherrod. "Adaptive Disk Spin-Down for Mobile Computers," *ACM/Baltzer Mobile Networks and Applications Journal*, vol. 5, no. 4 (2000), pp. 285–297.

24. Benjamin C. Reed, Darrell D. E. Long, Edward G. Chron and Randal C. Burns. "Authenticating Network Attached Storage," *IEEE Micro*, vol. 20, no. 1, January 2000, pp. 49–57.

25. Steven W. Carter and Darrell D. E. Long. "Improving Bandwidth Efficiency of Video-on-Demand Servers," *Computer Networks*, Elsevier, vol. 31, nos. 1–2, January 1999, pp. 99–111.

26. Eric C. Rosen, Theodore R. Haining, Darrell D. E. Long, Patrick E. Mantey. "REINAS: A Real-time System for Managing Environmental Data," *Journal of Software Engineering and Knowledge Engineering*, vol. 8, no. 1 (1998), pp. 35–53.

27. Darrell D. E. Long, Bruce R. Montague and Luis-Felipe Cabrera. "Swift/RAID: A Distributed RAID System," *Computing Systems*, vol. 7, no. 3 (1994), pp. 333–359.

28. Ivan Fellner, Ivan Racko, Milos Racek, Karol Fabian and Darrell D. E. Long. "A Comparison of Two Implementations of the Token Ring Priority Function," *International Journal in Computer Simulation*, vol. 5, no. 1 (1994), pp. 13–28.

29. Luis-Felipe Cabrera and Darrell D. E. Long. "Swift: Using Distributed Disk Striping to Provide High I/O Data Rates," *Computing Systems*, vol. 4, no. 4 (1991), pp. 405–436.

30. Richard A. Golding and Darrell D. E. Long. "Accessing Replicated Data in a Large-Scale Distributed System," *International Journal in Computer Simulation*, vol. 1, no. 4 (1991), pp. 347–372.

31. Jehan-François Pâris and Darrell D. E. Long. "The Performance of Available Copy Protocols for the Management of Replicated Data," *Performance Evaluation*, vol. 11, no. 1, April 1990, pp. 9–30.

32. Darrell D. E. Long, John L. Carroll and Kris Stewart. "Estimating the Reliability of Regeneration-Based Replica Control Protocols," *IEEE Transactions on Computers*, vol. 38, no. 12, (December 1989), pp. 1691–1702.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 99 of 411

Peer Reviewed Conference Papers

1. Eugene Chou, Leo Conrad-Shah, Austen Barker, Andrew Quinn, Ethan L. Miller and Darrell D. E. Long. "Lethe: Secure Deletion by Addition," *Proceedings of the Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems* (CHEOPS), Rome, ACM: Mary 8, 2023.

2. Kyle Fredrickson, Austen Barker and Darrell Long. "A Multiple Snapshot Attack on Deniable Storage Systems," *Proceedings of the Twenty-ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2021), IEEE, November 3–5, 2021.

3. Oceane Bel, Kenneth Chang, Nathan R. Tallent, Dirk Duellmann, Ethan L. Miller, Faisal Nawab and Darrell D. E. Long. "Geomancy: Automated Performance Enhancement through Data Layout Optimization," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, October 26–30, 2020.

4. James Byron, Ethan L. Miller and Darrell D. E. Long. "Measuring the Cost of Reliability in Archival Systems," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, October 26–30, 2020.

5. Austen Barker, Yash Gupta, Sabrina Au, Eugene Chou, Ethan L. Miller and Darrell D. E. Long. "Artifice: Data in Disguise," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, October 26–30, 2020.

6. Daniel Bittman, Peter Alvaro, Darrell Long and Ethan Miller. "Twizzler: a Data-Centric OS for Non-Volatile Memory," *Proceedings of the 2020 USENIX Annual Technical Conference,* Boston: Usenix Association, July 15–17, 2020.

7. Daniel Bittman, Peter Alvaro, Darrell D. E. Long and Ethan L. Miller. "A Tale of Two Abstractions: The Case for Object Space," *Proceedings of the Eleventh Hot Topics in Storage and File Systems* (Hot Storage 2019), Renton: Usenix Association, July 8–9, 2019.

8. Daniel Bittman, Darrell D. E. Long, Peter Alvaro, and Ethan L. Miller. "Optimizing Systems for Byte-Addressable NVM by Reducing Bit Flipping," *Proceedings of the Conference on File and Storage Technologies* (FAST), San Jose: Usenix Association, February 25–28, 2019.

9. Sinjoni Mukhopadhyay, Joel Frank, Daniel Bittman, Darrell Long and Ethan Miller. "Efficient Reconstruction Techniques for Disaster Recovery in Secret-Split Datastores," *Proceedings of the Twenty-sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2018), Milwaukee: IEEE, September 25–28, 2018.

10. James Byron, Ethan Miller and Darrell Long. "Using Simulation to Design Scalable and Cost-Efficient Archival Storage Systems," *Proceedings of the Twenty-sixth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2018), Milwaukee: IEEE, September 25–28, 2018.

11. Daniel Bittman, Matthew Gray, Justin Raizes, Sinjoni Mukhopadhyay, Matthew Bryson, Peter Alvaro, Darrell Long and Ethan Miller. "Designing Data Structures to Minimize Bit Flips on NVM," *Proceedings of the Seventh IEEE Non-Volatile Memory Systems and Applications Symposium* (NVMSA '18) Hakodate, Japan: IEEE, August 28–31, 2018.

12. Oceane Bel, Kenneth Chang, Daniel Bittman, Ethan L. Miller, Darrell D. E. Long and Hiroshi Isozaki. "Inkpack: A Secure, Data-Exposure Resistant Storage System," *Proceedings of the Eleventh Annual International Systems and Storage Conference* (SYSTOR), Haifa, Israel: ACM, June 4–8, 2018.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 100 of 411

13. Yan Li, Kenneth Chang, Oceane Bel, Ethan L. Miller and Darrell D. E. Long. "CAPES: Unsupervised Storage Performance Tuning Using Neural Network-Based Deep Reinforcement Learning," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (SC '17), November 13–16, 2017.

14. Carlos A. Rincon, Jehan-François Pâris, Ricardo Vilalta, Albert M. K. Cheng and Darrell D. E. Long. "Disk Failure Prediction in Heterogeneous Environments," *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems,* Seattle: IEEE, July 9–12, 2017.

15. Thomas Schwarz, Ahmed Amer, Thomas Kroeger, Ethan Miller, Darrell Long and Jehan-François Pâris. "RESAR: Reliable Storage at Exabyte Scale," *Proceedings of the Twenty-fourth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2016), London: IEEE, September 21–23, 2016.

16. Yan Li, Ethan Miller, Darrell Long and Yash Gupta. "Pilot: A Framework that Understands How to Do Performance Benchmarks the Right Way," *Proceedings of the Twenty-fourth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2016), London: IEEE, September 21–23, 2016.

17. Jehan-François Pâris and Darrell D. E. Long. "Pirogue: A lighter dynamic version of the Raft distributed consensus algorithm," *Proceedings of the Thirty-fourth International Performance of Computers and Communication Conference* (IPCCC 2015), Nanjing, China: IEEE, December 14–16, 2015.

18. Thomas Schwarz, Darrell D. E. Long and Jehan-François Pâris. "Triple failure tolerant storage systems using only exclusive-or parity calculations," *Proceedings of the Twenty-first IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC 2015), Zhangjiajie, China: IEEE, November 18–20, 2015.

19. Jehan-François Pâris and Darrell Long. "Reducing the Energy Footprint of a Distributed Consensus Algorithm," *Proceedings of the European Dependable Computing Conference* (EDCC 2015), Paris: IEEE, September 7–11, 2015.

20. Stephanie N. Jones, Ahmed Amer, Ethan L. Miller, Darrell D. E. Long, Rekha Pitchumani and Christina R. Strong. "Classifying Data to Reduce Long Term Data Movement in Shingled Write Disks," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, May 30–June 5, 2015.

21. Joel C. Frank, Shayna M. Frank, Lincoln A. Thurlow, Thomas M. Kroeger, Ethan L. Miller and Darrell D. E. Long. "Percival: A Searchable Secret Split Datastore," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, May 30–June 5, 2015.

22. Yan Li, Xiaoyuan Lu, Ethan L. Miller and Darrell D. E. Long. "ASCAR: Automating Contention Management for High-Performance Storage Systems," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Santa Clara University: IEEE, May 30–June 5, 2015.

23. Jehan-François Pâris, Ahmed Amer, Darrell Long and Thomas Schwarz. "Self-Repairing Disk Arrays," *Proceedings of the Fifth International Workshop on Adaptive Self-tuning Computing Systems* (ADAPT), January 21, 2015, Amsterdam, The Netherlands.

24. Avani Wildani, Ethan L. Milleri, Ian F. Adams and Darrell D. E. Long. "PERSES: Data Layout for Low Impact Failures," *Proceedings of the Twenty-second International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2014), Paris: IEEE, September 9–11, 2014.

25. Yan Li and Darrell D. E. Long. "Which Storage Device is Greenest? Modeling the Energy Cost of I/O Workloads," *Proceedings of the Twenty-second International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2014), Paris: IEEE, September 9–11, 2014.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 101 of 411

26. Jehan-François Pâris. Thomas J. E. Schwarz, Ahmed Amer and Darrell Long. "Protecting RAID Arrays Against Unexpectedly High Disk Failure Rates," *Proceedings of the Twentieth IEEE Pacific Rim International Symposium on Dependable Computing* (PRDC 2014), Singapore: IEEE, November 18–21, 2014.

27. Alex Nelson, Erik Steggall and Darrell Long. "Cooperative mode: Comparing interpretations of the Xbox 360 storage system," *Proceedings of the Digital Forensics Research Conference* (DFRWS), Denver: ACM, August 3–6, 2014.

28. Soror Sahri, Rim Moussa, Salima Benbernou and Darrell Long. "DBaaS-Expert: A Recommender for the Selection of the Right Cloud Database," *Proceedings of the Twenty-first International Symposium on Methodologies for Intelligent Systems* (ISMIS), Roskilde, Denmark: Springer, June 25–27, 2014.

29. Aleatha Parker-Wood, Darrell D. E. Long, Ethan Miller, Philippe Rigaux and Andy Isaacson. "A File By Any Other Name: Managing File Names with Metadata," *Proceedings of the Seventh Annual International Systems and Storage Conference* (SYSTOR), Haifa, Israel: ACM, June 10–12, 2014.

30. Ahmed Amer, Darrell D. E. Long and Thomas Schwarz. "Reliability Challenges for Storing Exabytes," *Proceedings of the International Conference on Computing, Networking and Communications* (ICNC), Honolulu: IEEE, February 3–6, 2014, pp. 907–913.

31. Jehan-François Pâris, Darrell D. E. Long and Thomas Schwarz, S.J. "Zero-Maintenance Disk Arrays," *Proceedings of the IEEE Nineteenth Pacific Rim International Symposium on Dependable Computing* (PRDC), Vancouver, British Columbia, Canada: IEEE, December 2–4, 2013.

32. Thomas Schwarz, Darrell D. E. Long and Jehan-François Pâris. "Reliability of Disk Arrays with Double Parity," *Proceedings of the IEEE Nineteenth Pacific Rim International Symposium on Dependable Computing* (PRDC), Vancouver, British Columbia, Canada: IEEE, December 2–4, 2013.

33. Jehan-François Pâris, Darrell D. E. Long and Witold Litwin. "Three-Dimensional Redundancy Codes for Archival Storage," *Proceedings of the Twenty-first International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS 2013), San Francisco: IEEE, August 14–16, 2013.

34. Aleatha Parker-Wood, Brian Madden, Michael M^{c}Throw, Darrell D.E. Long, Ian F. Adams and Avani Wildani. "Examining Extended and Scientific Metadata for Scalable Index Designs," *Proceedings of the Sixth Annual International Systems and Storage Conference* (SYSTOR), Haifa, Israel: ACM, June 10–12, 2013.

35. Hsu-Wan Kao, Jehan-François Pâris, Thomas Schwarz, S.J. and Darrell D. E. Long. "A Flexible Simulation Tool for Estimating Data Loss Risks in Storage Arrays," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Long Beach: IEEE, May 9, 2013.

36. Yan Li, Nakul Sanjay Dhotre, Yasuhiro Ohara, Thomas M. Kroeger, Ethan L. Miller and Darrell D. E. Long. "Horus: Fine-Grained Encryption-Based Security for Large-Scale Storage," *Proceedings of the Conference on File and Storage Technologies* (FAST), San Jose: Usenix Association, February 17–20, 2013.

37. Thomas Schwarz, Ignacio Corderí, Darrell D. E. Long and Jehan-François Pâris. "Simple, Exact Placement of Data in Containers," *Proceedings of International Conference on Computing, Networking and Communications* (ICNC), San Diego: IEEE, January 28–31, 2013.

38. Zhike Zhang, Deepavali Bhagwat, Witold Litwin, Darrell Long and Thomas Schwarz, S.J. "Improved Deduplication through Parallel Binning," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Austin: IEEE, December 1–3, 2012.

39. Jehan-François Pâris, Thomas Schwarz, S.J., Ahmed Amer and Darrell D. E. Long. "Highly Reliable Two-Dimensional RAID Arrays for Archival Storage," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Austin: IEEE, December 1–3, 2012.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 102 of 411

40. Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Yan Li, Darrell D. E. Long, Zhipeng Tan and Lei Chen. "A Hybrid Approach for Efficient Provenance Storage," *Proceedings of International Conference on Information and Knowledge Management* (CIKM), Maui: ACM, October 29–November 2, 2012.

41. Rekha Pitchumani, Andy Hospodor, Ahmed Amer, Yangwook Kang, Ethan L. Miller and Darrell Long. "Emulating a Shingled Write Disk," *Proceedings of the Twentieth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Washington, DC: IEEE, August 7–9, 2012.

42. Yan Li, Ethan L. Miller and Darrell D. E. Long. "Understanding Data Survivability in Archival Storage Systems," *Proceedings of the Fifth Annual International Systems and Storage Conference* (SYSTOR), Haifa, Israel: ACM, June 2012.

43. Witold Litwin, Darrell Long and Thomas Schwarz. "Combining Chunk Boundary and Chunk Signature Calculations for Deduplication," *Proceedings of the Tenth International Information and Telecommunication Technologies Symposium* (I2TS), Florianópolis, Brazil: IEEE, December 2011.

44. Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Dan Feng, Darrell D. E. Long, Yangwook Kang, Zhongying Niu and Zhipeng Tan. "Design and Evaluation of Oasis: An Active Storage Framework based on T10 OSD Standard," *Proceedings of the Conference on Mass Storage Systems and Technologies* (MSST), Denver, Colorado: IEEE, May 23–27, 2011.

45. Jehan-François Pâris, Ahmed Amer and Darrell Long. "Accelerated Chaining: A Better Way to Harness Peer Power in Video-on-Demand Applications," *Proceedings of the Twenty-sixth ACM Symposium on Applied Computing* (SAC 2011), Taichung, Taiwan: ACM, March 2011.

46. Thomas J. E. Schwarz and Darrell Long. "Clasas: A Key-Store for the Cloud," *Proceedings of the Eighteenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Miami Beach: IEEE, August 17–19, 2010.

47. Jehan-François Pâris, Thomas J. E. Schwarz, Ahmed Amer and Darrell Long. "Improving Disk Array Reliability Through Expedited Scrubbing," *Proceedings of the Fifth IEEE International Conference on Networking, Architecture, and Storage* (NAS 2010), Macau: IEEE, July 2010, pp. 119–125.

48. Aleatha Parker-Wood, Christina Strong, Ethan L. Miller and Darrell D. E. Long. "Security Aware Partitioning for Efficient File System Search," *Proceedings of the Conference on Mass Storage Systems and Technologies,* Incline Village, Nevada: IEEE, May 2010.

49. Ahmed Amer, Darrell D. E. Long, Ethan L. Miller, Jehan-François Pâris and Thomas J. E. Schwarz. "Design Issues for a Shingled Write Disk System," *Proceedings of the Conference on Mass Storage Systems and Technologies,* Incline Village, Nevada: IEEE, May 2010.

50. Jehan-François Pâris, Ahmed Amer, Darrell D. E. Long and Thomas J. E. Schwarz. "Evaluating the Impact of Irrecoverable Read Errors on Disk Array Reliability," *Proceedings of the IEEE Fifteenth Pacific Rim International Symposium on Dependable Computing* (PRDC), Shanghai, China: IEEE, November 2009.

51. Jehan-François Pâris, Ahmed Amer and Darrell D. E. Long. "Using Storage Class Memories to Increase the Reliability of Two-dimensional RAID Arrays," *Proceedings of the Seventeenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), London: IEEE, September 21–23, 2009.

52. Avani Wildani, Thomas J. E. Schwarz, Ethan L. Miller and Darrell D. E. Long. "Protecting Against Rare Event Failures in Archival Systems," *Proceedings of the Seventeenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), London: IEEE, September 21–23, 2009.

53. Deepavali Bhagwat, Kave Eshghi, Darrell D. E. Long and Mark Lillibridge. "Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup," *Proceedings of the Seventeenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), London: IEEE, September 21–23, 2009, pp. 237–245.

54. Kevin M. Greenan, Darrell D. E. Long, Ethan L. Miller, Thomas J. E. Schwarz and Avani Wildani. "Building Flexible, Fault-Tolerant Flash-based Storage Systems," *Proceedings of Hot Topics in System Dependability* (HotDep), Lisbon: IEEE, June 29, 2009.

55. Ahmed Amer, Jehan-François Pâris, Darrell D. E. Long and Thomas Schwarz. "Progressive Parity-Based Hardening of Data Stores," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Austin: IEEE, December 7–9, 2008, pp. 34–42.

56. Jehan-François Pâris, Thomas Schwarz, Darrell D. E. Long and Ahmed Amer. "When MTTDLs Are Not Good Enough: Providing Better Estimates of Disk Array Reliability," *Proceedings of the Seventh International Information and Telecommunication Technologies Symposium* (I2TS), Cuiabá, Brazil, Foz do Iguaçu, Paraná State, Brazil, December 3–5, 2008, pp. 190–195.

57. Kevin M. Greenan, Darrell D. E. Long, Ethan L. Miller, Thomas J. E. Schwarz, S.J. and Jay J. Wylie. "A Spin-Up Saved is Energy Earned: Achieving Power-Efficient, Erasure-Coded Storage," *Proceedings of Hot Topics in System Dependability* (HotDep), San Diego: Usenix Association, December 7, 2008.

58. Mark W. Storer, Kevin Greenan, Darrell D. E. Long and Ethan L. Miller. "Secure Data Deduplication," *Proceedings of the Fourth International Workshop on Storage Security and Survivability* (StorageSS), held in conjunction with the *ACM Computer and Communications Security Conference* (CCS), Alexandria, Virginia: ACM, October 31, 2008.

59. Ahmed Amer, Darrell D. E. Long, Jehan-François Pâris and Thomas Schwarz. "Increased Reliability with SSPiRAL Data Layouts," *Proceedings of the Sixteenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Baltimore: IEEE, September 8–10, 2008, pp. 189–198.

60. Kevin Greenan, Ethan L. Miller, Thomas Schwarz and Darrell D. E. Long. "Disaster Recovery Codes: Increasing Reliability with Large-Stripe Error Correction Codes," *Proceedings of the Third International Workshop on Storage Security and Survivability* (StorageSS), held in conjunction with the *ACM Computer and Communications Security Conference* (CCS), Alexandria, Virginia: ACM, October 29, 2007.

61. Kristal T. Pollack, Darrell D. E. Long, Richard A. Golding, Ralph A. Becker-Szendy and Benjamin Reed. "Quota Enforcement for High-performance Distributed Storage Systems," *Proceedings of the Conference on Mass Storage Systems and Technologies,* San Diego: IEEE, September 2007, pp. 72–84.

62. Timothy Bisson, Scott A. Brandt and Darrell D. E. Long. "A Hybrid Disk-Aware Spin-Down Algorithm with I/O Subsystem Support," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), New Orleans: IEEE, April 11–13, 2007.

63. Jehan-François Pâris, Thomas J. Schwarz, S.J. and Darrell D. E. Long. "Self-Adaptive Two-Dimensional RAID Arrays," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), New Orleans: IEEE, April 11–13, 2007.

64. Jehan-François Pâris, Thomas J. Schwarz, S.J. and Darrell D. E. Long. "Protecting Data Against Early Disk Failures," *Proceedings of the Fifth International Information and Telecommunication Technologies Symposium* (I2TS), Cuiabá, Brazil, December 2006, pp. 113–119.

65. Jehan-François Pâris, Thomas J. Schwarz, S.J. and Darrell D. E. Long. "Self-Adaptive Disk Arrays," *Proceedings of the Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems* (SSS), Springer, Lecture Notes in Computer Science: Dallas, November 2006, pp. 469–483.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 104 of 411

66. Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long and Carlos Maltzahn. "Ceph: A Scalable, High-Performance Distributed File System," *Proceedings of the Seventh Symposium on Operating Systems Design and Implementation* (OSDI), Seattle: Usenix Association, November 2006, pp. 307–320.

67. Timothy Bisson, Scott A. Brandt and Darrell D. E. Long. "NVCache: Increasing the Effectiveness of Disk Spin-down Algorithms with Caching," *Proceedings of the Fourteenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Monterey: IEEE, September 2006, pp. 422–432.

68. Jeffrey P. Rybczynski, Darrell D. E. Long and Ahmed Amer. "Adapting Predictions and Workloads for Power Management," *Proceedings of the Fourteenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Monterey: IEEE, September 2006, pp. 3–12.

69. Deepavali Bhagwat, Kristal Pollack, Darrell D. E. Long, Ethan L. Miller, Thomas Schwarz, S.J. and Jehan-François Pâris. "Providing High Reliability in a Minimum Redundancy Archival Storage System," *Proceedings of the Fourteenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Monterey: IEEE, September 2006, pp. 412–421.

70. Lawrence L. You, Kristal T. Pollack and Darrell D. E. Long. "Deep Store: an Archival Storage System Architecture," *Proceedings of the Twenty-first International Conference on Data Engineering* (ICDE), Tokyo: IEEE, April 2005, pp. 804–815.

71. Qin Xin, Ethan L. Miller, Thomas J. E. Schwarz, and Darrell D. E. Long. "Impact of Failure on Interconnection Networks in Large Storage Systems," *Proceedings of the Twenty-second Conference on Mass Storage Systems and Technologies,* Monterey: IEEE, April 2005, pp. 189–196.

72. Feng Wang, Bo Hong, Scott A. Brandt and Darrell D. E. Long. "Using MEMS-based Storage to Boost Disk Performance," *Proceedings of the Twenty-second Conference on Mass Storage Systems and Technologies,* Monterey: IEEE, April 2005, pp. 202–209.

73. Thomas J. E. Schwarz, Qin Xin, Ethan L. Miller, Darrell D. E. Long, Andrew Hospodor and Spencer Ng. "Disk Scrubbing in Large Archival Storage Systems," *Proceedings of the Twelfth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Volendam, The Netherlands: IEEE, October 2004, pp. 409–418.

74. Karl Brandt, Darrell D. E. Long and Ahmed Amer. "Predicting When Not To Predict," *Proceedings of the Twelfth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Volendam, The Netherlands: IEEE, October 2004, pp. 419–426.

75. Bo Hong, Thomas J. E. Schwarz, Scott A. Brandt and Darrell D. E. Long. "Reliability of MEMS-Based Storage Enclosures," *Proceedings of the Twelfth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Volendam, The Netherlands: IEEE, October 2004, pp. 571–579.

76. Bo Hong, Demyn Plantenberg, Darrell D. E. Long and Miriam Sivan-Zimet. "Duplicate Data Elimination in a SAN File System," *Proceedings of the Twenty-first Symposium on Mass Storage Systems* (MSS), Goddard, Maryland: NASA, April 2004, pp. 301–314.

77. Purvi Shah, Jehan-François Pâris, Ahmed Amer and Darrell D. E. Long. "Identifying Stable File Access Patterns," *Proceedings of the Twenty-first Symposium on Mass Storage Systems* (MSS), Goddard, Maryland: NASA, April 2004, pp. 159–163.

78. Feng Wang, Qin Xin, Bo Hong, Scott A. Brandt, Ethan Miller, Darrell Long and Tyce T. M^{c}Larty. "File System Workload Analysis For Large Scale Scientific Computing Applications," *Proceedings of the Twenty-first Symposium on Mass Storage Systems* (MSS), Goddard, Maryland: NASA, April 2004, pp. 139–152.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 105 of 411

79. Feng Wang, Scott A. Brandt, Ethan L. Miller and Darrell D. E. Long. "OBFS: A File System for Object-Based Storage Devices," *Proceedings of the Twenty-first Symposium on Mass Storage Systems* (MSS), Goddard, Maryland: NASA, April 2004, pp. 283–300.

80. Ismail Ari, Bo Hong, Ethan L. Miller, Scott A. Brandt and Darrell D. E. Long. "Managing Flash Crowds on the Internet," *Proceedings of the Eleventh International Symposium on Modeling, Analysis, and Simulation in Computer and Telecommunication Systems* (MASCOTS), Orlando: IEEE, October 2003, pp. 246–249.

81. Bo Hong, Scott A. Brandt, Darrell D. E. Long, Ethan L. Miller, Karen A. Glocer and Zachary N. J. Peterson. "Zone-Based Shortest Positioning Time First Scheduling for MEMS-Based Storage Devices," *Proceedings of the Eleventh International Symposium on Modeling, Analysis, and Simulation in Computer and Telecommunication Systems* (MASCOTS), Orlando: IEEE, October 2003, pp. 104–113.

82. Karthik Thirumalai, Jehan-François Pâris and Darrell D. E. Long. "Tabbycat: An Inexpensive Scalable Server for Video-on-Demand," *Proceedings of the IEEE 2003 International Conference on Communications* (ICC), Anchorage, Alaska: IEEE, May 2003, pp. 896–900.

83. Ahmed Amer, Alison Luo, Newton Der, Darrell D. E. Long and Alex Pang. "Visualizing Cache Effects on I/O Workload Predictability," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2003, pp. 417–424.

84. Jehan-François Pâris and Darrell D. E. Long. "A Proactive Implementation of Interactive Video-on-Demand," *Proceedings of the International Performance, Computing, and Communications Conference* (IPCCC), Phoenix: IEEE, April 2003, pp. 425–431.

85. Qin Xin, Ethan L. Miller, Scott A. Brandt, Darrell D. E. Long, Thomas Schwarz and Witold Litwin. "Reliability Mechanisms for Very Large Storage Systems," *Proceedings of the Twentieth Symposium on Mass Storage Systems* (MSS), San Diego: IEEE, April 2003, pp. 146–156.

86. Gary A. S. Whittle, Jehan-François Pâris, Ahmed Amer, Darrell D. E. Long and Randal Burns. "Using Multiple Predictors to Improve the Accuracy of File Access Predictions," *Proceedings of the Twentieth Symposium on Mass Storage Systems* (MSS), San Diego: IEEE, April 2003, pp. 230–240.

87. Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long and Lan Xue. "Efficient Metadata Management in a Large Distributed Storage System," *Proceedings of the Twentieth Symposium on Mass Storage Systems* (MSS), San Diego: IEEE, April 2003, pp. 290–298.

88. Jehan-François Pâris and Darrell D. E. Long. "A Variable Bandwidth Broadcasting Protocol For Video-on-Demand," *Proceedings of the 2003 SPIE Multimedia Computing and Networking Conference* (MMCN), San Jose: SPIE, January 2003, pp. 209–219.

89. Ying Lin, Scott A. Brandt, Darrell D. E. Long and Ethan L. Miller. "Power Conservation Strategies for MEMS-based Storage Devices," *Proceedings of the Tenth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Fort Worth: IEEE, October 2002, pp. 53–62.

90. Ahmed Amer, Darrell D. E. Long and Randal C. Burns. "Group-Based Management of Distributed File Caches," *Proceedings of the Twenty-second International Conference on Distributed Computing Systems* (ICDCS), Vienna: IEEE, July 2002, pp. 525–534.

91. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Increasing Predictive Accuracy by Prefetching Multiple Program and User Specific Files," *Proceedings of the Sixteenth Annual International Symposium on High Performance Computing Systems and Applications* (HPCS), Moncton, New Brunswick, Canada: IEEE, June 2002, pp. 12–19.

92. Alison Luo, Ahmed Amer, Scott Speidel, Darrell D. E. Long and Alex Pang. "Visualizing I/O Predictability," *Proceedings of the First International Symposium on 3D Data Processing, Visualization, and Transmission* (3DPVT), Padova, Italy: IEEE, June 2002, pp. 202–207.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 106 of 411

93. Ahmed Amer, Darrell D. E. Long, Jehan-François Pâris and Randal C. Burns. "File Access Prediction with Adjustable Accuracy," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2002, pp. 131–140.

94. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Increasing Predictive Accuracy through Limited Prefetching," *Proceedings of Communications Networks and Distributed Systems Modeling and Simulation* (CNDS), San Antonio: Society for Computer Simulation, January 2002, pp. 131–138.

95. Ethan L. Miller, Darrell D. E. Long, William Freeman and Benjamin C. Reed. "Strong Security for Network-Attached Storage," *Proceedings of the Conference on File and Storage Technologies* (FAST), Monterey: Usenix Association, January 2002, pp. 1–13.

96. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Using Program and User Information to Improve File Prediction Performance," *Proceedings of the International Symposium on Performance Analysis of Systems and Software* (ISPASS), Tucson: IEEE, November 2001, pp. 111–119.

97. Ahmed Amer and Darrell D. E. Long. "Aggregating Caches: A Mechanism for Implicit File Prefetching," *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Cincinnati: IEEE, August 2001, pp. 293–301.

98. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Performing File Prediction with a Program-Based Successor Model," *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (MASCOTS), Cincinnati: IEEE, August 2001, pp. 193–202.

99. Thomas M. Kroeger and Darrell D. E. Long. "Design and Implementation of a Predictive File Prefetching Algorithm," *Proceedings of Usenix Technical Conference,* Boston: Usenix Association, June 2001, pp. 105–118.

100. Ethan L. Miller, Scott A. Brandt and Darrell D. E. Long. "HeRMES: High-Performance Reliable MRAM-Enabled Storage," *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems* (HotOS-VIII), Elmau, Germany: IEEE, May 2001, pp. 95–99.

101. Ahmed Amer and Darrell D. E. Long. "Noah: Low-cost File Access Prediction Through Pairs," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2001, pp. 27–33.

102. Ethan Miller, Darrell Long, William Freeman and Benjamin Reed. "Strong Security for Distributed File Systems," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2001, pp. 34–40.

103. Scott R. Carter, Jehan-François Pâris, Saurabh Mohan and Darrell D. E. Long. "A Dynamic Heuristic Broadcasting Protocol for Video-on-Demand," *Proceedings of the Twenty-first International Conference on Distributed Computing Systems* (ICDCS), Mesa, Arizona: IEEE, April 2001, pp. 657–664.

104. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "An Analytical Study of Opportunistic Lease Renewal," *Proceedings of the Twenty-first International Conference on Distributed Computing Systems* (ICDCS), Mesa, Arizona: IEEE, April 2001, pp. 146–153.

105. Steven W. Carter, Darrell D. E. Long and Jehan-François Pâris. "Efficient Implementation of Interactive Video-on-Demand," *Proceedings of the Eighth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), San Francisco: IEEE, August–September 2000, pp. 172–179.

106. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "Consistency and Locking for Distributing Updates to Web Servers Using a File System," *Proceedings of Performance and Architecture of Web Servers* (PAWS), Santa Clara: ACM, June, 2000 and *Performance Evaluation Review,* vol. 28, no. 2, September 2000.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 107 of 411

107. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "Safe Caching in a Distributed File System for Network Attached Storage," *Proceedings of the International Parallel and Distributed Processing Symposium* (IPDPS), Cancun: IEEE, May 2000, pp. 155–162.

108. Jehan-François Pâris, Theodore R. Haining and Darrell D. E. Long. "A Stack Model Based Replacement Policy for a Non-Volatile Write Cache," *Proceedings of the Eighth NASA Goddard Conference on Mass Storage Systems and Technologies,* College Park: IEEE, March 2000, pp. 217–223.

109. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "Semi-Preemptible Locks for a Distributed File System," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, February 2000, pp. 397–404.

110. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "A Reactive Broadcasting Protocol for Video on Demand," *Proceedings of the Multimedia Computing and Networking Conference* (MMCN), San Jose: SPIE, January 2000, pp. 216–223.

111. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "Combining Pay-per-View and Video-on-Demand Services," *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), College Park: IEEE, October 1999, pp. 270–276.

112. Jehan-François Pâris, Darrell D. E. Long and Patrick E. Mantey. "A Zero-delay Broadcasting Protocol for Video on Demand," *Proceedings of the Seventh ACM International Multimedia Conference,* Orlando: ACM, October 1999, pp. 189–197.

113. Benjamin C. Reed, Darrell D. E. Long, Edward G. Chron and Randal C. Burns. "Authenticating Network Attached Storage," *Proceedings of the Symposium on High Performance Interconnects* (Hot Interconnects VII), Stanford: IEEE, August 1999, pp. 49–57.

114. Thomas M. Kroeger and Darrell D. E. Long. "The Case for Efficient File Access Pattern Modeling," *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems* (HotOS-VII), Rio Rico, Arizona: IEEE, March 1999, pp. 14–19.

115. Theodore R. Haining and Darrell D. E. Long. "Management Policies for Non-volatile Write Caches," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, February 1999, pp. 321–328.

116. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "A Hybrid Broadcasting Protocol for Video on Demand," *Proceedings of the Multimedia Computing and Networking Conference* (MMCN), San Jose: SPIE, January 1999, pp. 317–326.

117. Timothy Gibson, Ethan L. Miller and Darrell D. E. Long. "Long-term File Activity and Inter-reference Patterns," *Proceedings of the Computer Measurement Group Conference,* Anaheim: CMG, December 1998, pp. 976–987.

118. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "A Low Bandwidth Broadcasting Protocol for Video on Demand," *Proceedings of the International Conference on Computer Communication and Networks* (ICCCN), Lafayette: IEEE, October 1998, pp. 690–697.

119. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "Efficient Broadcasting Protocols for Video on Demand," *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), Montreal: IEEE, July 1998, pp. 127–130.

120. Randal C. Burns and Darrell D. E. Long. "In-Place Reconstruction of Delta Compressed Files," *Proceedings of Principles of Distributed Computing* (PODC), Puerto Vallarta: ACM, June 1998, pp. 267–275.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 108 of 411

121. Thomas M. Kroeger, Darrell D. E. Long and Jeffrey C. Mogul. "Exploring the Bounds of Web Latency Reduction from Caching and Prefetching," *Proceedings of the Symposium on Internet Technologies and Systems,* Monterey: Usenix Association, 1997, pp. 13–22.

122. Randal C. Burns and Darrell D. E. Long. "Efficient Distributed Back-up with Delta Compression," *Proceedings of I/O in Parallel and Distributed Systems* (IOPADS), San Jose: ACM, November 1997, pp. 27–36.

123. Steven W. Carter and Darrell D. E. Long. "Improving Video-on-Demand Server Efficiency Through Stream Tapping," *Proceedings of the Sixth International Conference on Computer Communications and Networks* (ICCCN), Las Vegas: IEEE, September 1997, pp. 200–207.

124. Darrell D. E. Long and Jehan-François Pâris. "Voting Without Version Numbers," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, February 1997, pp. 139–145.

125. Randal C. Burns and Darrell D. E. Long. "A Linear Time, Constant Space Differencing Algorithm," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, February 1997, pp. 429–436.

126. David P. Helmbold, Darrell D. E. Long and Bruce Sherrod. "A Dynamic Disk Spin-down Technique for Mobile Computing," *Proceedings of the Second Annual International Conference on Mobile Computing and Networking* (MobiCom), Rye, New York: ACM, November 1996, pp. 130–142.

127. Darrell D. E. Long and Jehan-François Pâris. "A Leaner, More Efficient, Available Copy Protocol," *Proceedings of the Symposium on Parallel and Distributed Processing* (SPDP), New Orleans: IEEE, October 1996, pp. 400–407.

128. Darrell D. E. Long, Patrick E. Mantey, Eric C. Rosen, Craig M. Wittenbrink and Bruce Gritton. "REINAS: A Real-time System for Managing Environmental Data," *Proceedings of the Eighth Software Engineering and Knowledge Engineering Conference,* Lake Tahoe: SEKE, June 1996, pp. 293–300.

129. Thomas M. Kroeger and Darrell D. E. Long. "Predicting File-System Actions from Prior Events," *Proceedings of 1996 Usenix Winter Technical Conference,* San Diego: Usenix Association, January 1996, pp. 319–328.

130. Darrell Long, Andrew Muir and Richard Golding. "A Longitudinal Study of Internet Host Reliability," *Proceedings of the Fourteenth Symposium on Reliable Distributed Systems* (SRDS), Bad Neuenahr, Germany: IEEE, September 1995, pp. 2–9.

131. Chane L. Fullmer, Darrell D. E. Long and Luis-Felipe Cabrera. "Adding Adaptive Flow Control to Swift/RAID," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, March 1995, pp. 290–296.

132. Darrell D. E. Long, Patrick E. Mantey, Theodore R. Haining, Bruce R. Montague, Eric C. Rosen. "An Archiecture Supporting Real-Time and Retrospective Environmental Data Management," *Proceedings of the First International Conference for Applications of Databases (ADB),* June 1994.

133. Cheng Tang, Shree Murthy, and Darrell D. E. Long. "Performance Guarantees on ATM Networks," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 1994, pp. 111–118.

134. Richard A. Golding, Darrell D. E. Long and John Wilkes. "The *refdbms* Distributed Bibliographic Database System," *Proceedings of the 1994 Usenix Winter Technical Conference,* San Francisco: Usenix Association, January 1994, pp. 47–62.

135. Richard A. Golding and Darrell D. E. Long. "Using an Object-oriented Framework to Construct Wide-area Group Communication Mechanisms," *Proceedings of the International Symposium on Applied Computing,* Monterrey, Mexico, October 1993, pp. 65–74.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 109 of 411

136. Darrell D. E. Long, Carol Osterbrock and Luis-Felipe Cabrera. "Providing Performance Guarantees in an FDDI Network," *Proceedings of the Thirteenth International Conference on Distributed Computing Systems* (ICDCS), Pittsburgh: IEEE, May 1993, pp. 328–336.

137. Darrell D. E. Long and Madhukar N. Thakur. "Scheduling Real-Time Disk Transfers for Continuous Media Applications," *Proceedings of the Twelfth Symposium on Mass Storage Systems* (MSS) Monterey: IEEE, April 1993, pp. 227–232.

138. Richard A. Golding and Darrell D. E. Long. "Simulation Modeling of Weak-Consistency Protocols," *Proceedings of the First International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), San Diego: Society for Computer Simulation, January 1993, pp. 233–238.

139. Perry K. Sloope, Jehan-François Pâris and Darrell D. E. Long. "A Simulation Study of Replication Control Protocols Using Volatile Witnesses," *Proceedings of the Twenty-fifth Annual Simulation Symposium,* Orlando: IEEE, April 1992, pp. 108–117.

140. Jehan-François Pâris and Darrell D. E. Long. "Protecting Replicated Objects Against Media Failures," *Proceedings of the Workshop on Research Issues on Transaction and Query Processing,* Phoenix: IEEE, February 1992, pp. 109–115.

141. Richard A. Golding and Darrell D. E. Long. "Quorum-oriented Multicast Protocols for Data Replication," *Proceedings of the Eighth International Conference on Data Engineering* (ICDE), Phoenix: IEEE, February 1992, pp. 490–497.

142. Luis-Felipe Cabrera and Darrell D. E. Long. "Swift: A Distributed Storage Architecture for Large Objects," *Proceedings of the Eleventh Symposium on Mass Storage Systems* (MSS), Monterey: IEEE, October 1991, pp. 123–128.

143. Darrell D. E. Long, John L. Carroll and C. J. Park. "A Study of the Reliability of Internet Sites," *Proceedings of the Tenth Symposium on Reliable Distributed Systems* (SRDS), Pisa: IEEE, September 1991, pp. 177–186.

144. Darrell D. E. Long and Luis-Felipe Cabrera. "Exploiting Multiple I/O Streams to Provide High Data-Rates," *1991 Summer Usenix Technical Conference,* Nashville: Usenix Association, June 1991, pp. 31–48.

145. Jehan-François Pâris and Darrell D. E. Long. "Voting with Regenerable Volatile Witnesses," *Proceedings of the Seventh International Conference on Data Engineering* (ICDE), Kobe, Japan: IEEE, April 1991, pp. 112–119.

146. Jehan-François Pâris and Darrell D. E. Long. "Resilient Memory-Resident Data Objects," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, March 1991, pp. 145–151.

147. Darrell D. E. Long, John L. Carroll and Kris Stewart. "The Reliability of Regeneration-Based Replica Control Protocols," *Proceedings of the International Conference on Distributed Computing Systems* (ICDCS), Newport Beach: IEEE, June 1989, pp. 465–473.

148. John L. Carroll and Darrell D. E. Long. "The Effect of Failure and Repair Distributions on Consistency Protocols for Replicated Data Objects," *Proceedings of the Twenty-second Annual Simulation Symposium,* Tampa: IEEE, March 1989, pp. 47–60.

149. Darrell D. E. Long, Jehan-François Pâris and John L. Carroll. "Reliability of Replicated Data Objects," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, March 1989, pp. 402–406.

150. Darrell D. E. Long and Jehan-François Pâris. "Regeneration Protocols for Replicated Objects," *Proceedings of the International Conference on Data Engineering* (ICDE), Los Angeles: IEEE, February 1989, pp. 538–545.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 110 of 411

151. Darrell D. E. Long and Jehan-François Pâris. "A Realistic Evaluation of Optimistic Dynamic Voting," *Proceedings of the Seventh Symposium on Reliable Distributed Systems* (SRDS), Columbus: IEEE, October 1988, pp. 129–137.

152. Jehan-François Pâris, Darrell D. E. Long and Alexander Glockner. "A Realistic Evaluation of Consistency Algorithms for Replicated Files," *Proceedings of the Twenty-first Annual Simulation Symposium,* Tampa: IEEE, March 1988, pp. 121–130.

153. Jehan-François Pâris and Darrell D. E. Long. "Efficient Dynamic Voting Algorithms," *Proceedings of the International Conference on Data Engineering* (ICDE), Los Angeles: IEEE, February 1988, pp. 268–275.

154. John L. Carroll, Darrell D. E. Long and Jehan-François Pâris. "Block-Level Consistency of Replicated Files," *Proceedings of the International Conference on Distributed Computing Systems* (ICDCS), Berlin: IEEE, September 1987, pp. 146–153.

155. Darrell D. E. Long and Jehan-François Pâris. "On Improving the Availability of Replicated Files," *Proceedings of the Symposium on Reliability in Distributed Software and Database Systems* (SRDSDS), Williamsburg: IEEE, March 1987, pp. 77–83.

Peer Reviewed Short Papers

1. Oceane Bel, Sinjoni Mukhopadhyay, Nathan Tallent, Faisal Nawab and Darrell Long. "WinnowML: Stable Feature Selection for Maximizing Prediction Accuracy of Time-based System Modeling," *IEEE International Conference on Big Data (Big Data)*, December 15–18, 2021, *to appear.*

2. Oceane Bel, Kenneth Chang, Nathan R. Tallent, Dirk Duellmann, Ethan L. Miller, Faisal Nawab and Darrell D. E. Long. "Geomancy: Automated Performance Enhancement through Data Layout Optimization," *Proceedings of the International Symposium on Performance Analysis of Systems and Software* (ISPASS), IEEE: Virtual, August 23–26, 2020.

3. Austen Barker, Staunton Sample, Yash Gupta, Anastasia McTaggart, Ethan L. Miller and Darrell D. E. Long "Artifice: A Deniable Steganographic File System," *Proceedings of Ninth Usenix Workshop on Free and Open Communications on the Internet,* (FOCI 2019), Usenix Association: Santa Clara, August 13, 2019.

4. Jehan-François Pâris, Thomas Schwarz, S. J. and Darrell D. E. Long. "Improving Disk Array Reliability Through Faster Repairs," *Proceedings of the Thirty-fifth International Performance of Computers and Communication Conference* (IPCCC 2016), Las Vegas, NV: IEEE, December 2016.

5. Daniel Bittman, D.J. Capelis and Darrell Long. "Introducing SeaOS," *Proceedings of the 2014 International Conference on Information Science and Applications* (ICISA), Seoul, Korea: IEEE, May 6–9, 2014, pp. 1–3.

6. Ranjana Rajendran, Ethan L. Miller and Darrell D. E. Long. "Horus: Fine-Grained Encryption-Based Security for High Performance Petascale Storage," *Proceedings of the Sixth Petascale Data Storage Workshop* (PDSW), Seattle: ACM, November 13, 2011.

7. Stephanie N. Jones, Christina R. Strong, Aleatha Parker-Wood, Alexandra Holloway and Darrell D. E. Long. "Easing the Burdens of HPC File Management," *Proceedings of the Fifth Petascale Data Storage Workshop* (PDSW), Seattle: ACM, November 13, 2011.

8. Yulai Xie, Kiran-Kumar Muniswamy-Reddy, Darrell D. E. Long, Ahmed Amer, Dan Feng and Zhipeng Tan. "Compressing Provenance Graphs," *Proceedings of the Workshop on the Theory and Practice of Provenance* (TaPP), Crete: Usenix Association, June 20–21, 2011.

9. Stephanie N. Jones, Christina R. Strong, Darrell D. E. Long and Ethan L. Miller. "Tracking Emigrant File Data via Transient Provenance," *Proceedings of the Workshop on the Theory and Practice of Provenance* (TaPP) Crete: Usenix Association, June 20–21, 2011.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 111 of 411

10. Ignacio Corderi, Thomas Schwarz, Ahmed Amer, Darrell D. E. Long and Jehan-François Pâris. "Self-Adjusting Two-Failure Tolerant Disk Arrays," *Proceedings of the Fifth Petascale Data Storage Workshop* (PDSW), New Orleans: ACM, November 15, 2010.

11. Sara Chaarawi, Jehan-François Pâris, Ahmed Amer, Thomas Schwarz and Darrell D. E. Long. "Using a Shared Storage Class Memory Device to Improve the Reliability of RAID Arrays," *Proceedings of the Fifth Petascale Data Storage Workshop* (PDSW), New Orleans: ACM, November 15, 2010.

12. D. J. Capelis and Darrell D. E. Long. "Fived: A Service-Based Architecture Implementation to Innovate at the Endpoints," *Proceedings of SIGCOMM 2010*, New Dehli, India: ACM, September 2010, pp. 419–420.

13. Ian F. Adams, Darrell D. E. Long and Ethan L. Miller. "Maximizing Efficiency By Trading Storage for Computation," *Proceedings of the Workshop on Hot Topics in Cloud Computing* (Hot Cloud), San Diego: Usenix Association, June 2009.

14. Mark W. Storer, Kevin M. Greenan, Ian F. Adams Ethan L. Miller, Darrell D. E. Long and Kaladhar Voruganti. "Logan: Automatic Management for Evolvable, Large-Scale, Archival Storage," *Proceedings of the 2008 Petascale Data Storage Workshop* (PDSW), IEEE, November 2008, pp. 1–8.

15. Neerja Bhatnagar, Kevin M. Greenan, Rosie Wacha, Ethan L. Miller and Darrell D. E. Long. "Energy-Reliability Tradeoffs in Sensor Network Storage," *Proceedings of Fifth Workshop on Embedded Networked Sensors* (HotEmNets), Charlottesville, Virginia: ACM, June 2–3, 2008.

16. Jehan-François Pâris, Darrell D. E. Long and Thomas J. E. Schwarz. "Ensuring Data Survival in Solid-State Storage Devices," *Proceedings of the First International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability* (SPEED), Salt Lake City: IEEE, February 2008.

17. Jehan-François Pâris and Darrell D. E. Long. "Using Device Diversity to Protect Data against Batch-Correlated Disk Failures," *Proceedings of the Second International Workshop on Storage Security and Survivability* (StorageSS), held in conjunction with the *ACM Computer and Communications Security Conference* (CCS), Alexandria, Virginia: ACM, October 30, 2006.

18. Jehan-François Pâris and Darrell D. E. Long. "An Analytic Study of Stream Tapping Protocols," *Proceedings of the 2006 IEEE International Conference on Multimedia* (ICME), Toronto: IEEE, July 2006, pp. 1237–1240.

19. Jeffrey P. Rybczynski, Darrell D. E. Long and Ahmed Amer. "Expecting the Unexpected: Adaptation for Predictive Energy Conservation," *Proceedings of the International Workshop on Storage Security and Survivability* (StorageSS), held in conjunction with the *ACM Computer and Communications Security Conference* (CCS), Alexandria, Virginia: ACM, November 2005, pp. 130–134.

20. Jehan-François Pâris, Ahmed Amer and Darrell D. E. Long. "A Stochastic Approach to File Access Prediction," *Proceedings of the International Workshop on Storage Network Architecture and Parallel I/O* (SNAPI), New Orleans: IEEE, September 2003.

21. Ismail Ari, Ahmed Amer, Robert Gramacy, Ethan Miller, Scott Brandt and Darrell Long. "Who is more Adaptive? ACME: Adaptive Caching Using Multiple Experts," *Proceedings of the 2002 Workshop on Distributed Data and Structures* (WDAS), Paris, France: Carleton Scientific, March 2002.

22. Jehan-François Pâris and Darrell D. E. Long. "The Case for Aggressive Partial Preloading in Video-on-Demand Broadcasting Protocols," *Proceedings of the 2001 IEEE International Conference on Multimedia* (ICME), Tokyo, Japan: IEEE, August 22–25, 2001, pp. 113–116.

23. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Caching Files with a Program-based Last *n* Successors Model," *Proceedings of the Workshop on Caching, Coherence and Consistency* (WC3), Sorrento, Italy: ACM, June 2001.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 112 of 411

24. Ahmed Amer and Darrell D. E. Long. "Adverse Filtering Effects and the Resilience of Aggregating Caches," *Proceedings of the Workshop on Caching, Coherence and Consistency* (WC3), Sorrento, Italy: ACM, June 2001.

25. Tsozen Yeh, Darrell D. E. Long and Scott A. Brandt. "Conserving Battery Energy through Making Fewer Incorrect File Predictions," *Proceedings of the IEEE Workshop on Power Management for Real-Time and Embedded Systems* (at the IEEE Real-Time Technology and Applications Symposium), Taipei, Taiwan: IEEE, May 2001, pp. 30–36.

26. Ahmed Amer and Darrell D. E. Long. "Dynamic Relationships and the Persistence of Pairings," *Proceedings of the International Workshop on Wireless Networks and Mobile Computing* (WNMC), Mesa, Arizona: IEEE, April 2001, pp. 502–507.

27. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "A Universal Distribution Protocol for Video-on-Demand," *Proceedings of the 2000 IEEE International Conference on Multimedia* (ICME), New York: IEEE, July 2000, vol. 1, pp. 49–52.

28. Jehan-François Pâris and Darrell D. E. Long. "Limiting the Client Bandwidth of Broadcasting Protocols for Videos on Demand," *Proceedings of Euromedia 2000,* Antwerp: Society for Computer Simulation, May 2000, pp. 107–111.

29. Jehan-François Pâris, Steven W. Carter and Darrell D. E. Long. "Bandwidth Allocation Issues in Near Video on Demand Services," *Proceedings of the Third Annual Multimedia Technology and Applications Conference* (MTAC), Anaheim: IEEE, September 1998, pp. 196–200.

30. Darrell D. E. Long. "A Replicated Monitoring Tool," *Proceedings of the Second Workshop on the Management of Replicated Data* (WMRD), Monterey: IEEE, November 1992, pp. 96–99.

31. Darrell D. E. Long. "Analysis of Replication Control Protocols," *Proceedings of the First Workshop on the Management of Replicated Data* (WMRD), Houston: IEEE, November 1990, pp. 117–122.

32. Jehan-François Pâris, Darrell D. E. Long and Walter A. Burkhard. "The Management of Consistency in Fault-Tolerant File Systems," *Proceedings of the Workshop on Fault Tolerance in Parallel and Distributed Computing,* San Diego: IEEE, December 1987, pp. 58–59.

33. Darrell D. E. Long. "Optimistic Algorithms for Replicated Data Management," *Proceedings of the Second Workshop on Large-Grained Parallelism,* Carnegie-Mellon University, November 1987, pp. 58–59.

Invited Articles

1. David Pease and Darrell D.E. Long. "Future File Systems," *Proceedings of Computing with Massive and Persistent Data* (CMPD), Baltimore: IEEE, September 22, 2008.

2. Randal Burns, Larry Stockmeyer and Darrell D.E. Long. "Experimentally Evaluating In-Place Delta Reconstruction," *Proceedings of the NASA and IEEE Mass Storage Conference,* College Park: IEEE, April 2002, pp. 137–151.

3. Darrell D. E. Long. "A Technique for Managing Mirrored Disks," *Proceedings of the International Performance Conference on Computers and Communication* (IPCCC), Phoenix: IEEE, April 2001, pp. 272–277.

4. Darrell D. E. Long, Patrick E. Mantey, Craig M. Wittenbrink, Theodore R. Haining and Bruce R. Montague. "REINAS: the Real-Time Environmental Information Network and Analysis System," *Proceedings of the IEEE Computer Society CompCon,* San Francisco: IEEE, March 1995, pp. 482–487.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 113 of 411

Miscellanea

1. Daniel Bittman, Peter Alvaro, Darrell D. E. Long and Ethan Miller. "The Flipside: A Bit Flip Saved Is Power and Lifetime Earned," *;login: The Usenix Magazine,* 2019, vol. 44, no. 2.

2. Ahmed Amer, Darrell D. E. Long, Ethan L. Miller, Jehan-Fraçois Pâris and Thomas Schwarz. "Data Management and Layout for Shingled Magnetic Recording," *IEEE International Magnetics Conference* (InterMag 2011), Taipei, Taiwan: IEEE, April 25–29, 2011.

3. K. Gopinath, Jon Elerath and Darrell Long. "Reliability Modelling of Disk Subsystems with Probabilistic Model Checking," *Dagstuhl Seminar on Quantitative and Qualitative Analysis of Network Protocols,* January 2010.

4. António M. Baptista, Michael Wilkin, Phillip Pearson, Paul Turner, Cole M^{c}Candlish, Phillip Barrett, Salil Das, Wendy Sommerfield, Ming Qi, Neetu Nangia, David Jay, Darrell Long, Calton Pu, John Hunt, Zhaoqing Yang, Edward Myers, Jeff Darland and Anna Farrenkopf, "Towards a Multipurpose Forecast Systems for the Columbia River Estuary," *Proceedings of the Ocean Community Conference,* Baltimore: Marine Technology Society, November 1998.

5. Benjamin Reed and Darrell D. E. Long, "Analysis of Caching Algorithms for Distributed File Systems," *Operating Systems Review,* July 1996, vol. 30, no. 3, pp. 12–21.

6. Daniel M. Fernandez, Patrick E. Mantey, Darrell D. E. Long, Eric Rosen and Craig M. Wittenbrink. "REINAS: Real-Time Environmental Information Network," *Sea Technology,* May 1996, vol. 37, no. 5, pp. 47–53.

7. Darrell D. E. Long. "A Note On Bit-Mapped Free Sector Management," *Operating Systems Review,* April 1993, vol. 27, no. 2, pp. 7–9.

Patents

1. Darrell D. E. Long and Thomas M. Kroeger. "Predictive Event Tracking Method," U.S. Patent no. 5,889,993, March 1999.

2. Luis Felipe Cabrera and Darrell D. E. Long. "Method and Apparatus for Establishing and Maintaining the Status of Membership Sets used in Mirrored Read and Write Input/Output without Logging," U.S. Patent no. 5,917,998, June 1999.

3. Randal C. Burns, Edward G. Chron, Darrell D. E. Long and Benjamin C. Reed. "Secure Array of Remotely Encrypted Storage Devices," U.S. Patent no. 5,931,947, August 1999.

4. Randal C. Burns and Darrell D. E. Long. "A Method for Generating and Reconstructing In-place Delta Files," U.S. Patent no. 6,018,747, January 2000.

5. Edward G. Chron, Darrell D. E. Long, Randal C. Burns and Benjamin Reed. "Decentralized Remotely Encrypted File System," U.S. Patent no. 6,405,315, June 2002.

6. Randal C. Burns, Robert M. Rees and Darrell D. E. Long. "Data Placement and Allocation using Virtual Contiguity," U.S. Patent no. 6,651,147, November 2003.

7. Randal C. Burns, Robert M. Rees, Darrell D. E. Long and Atul Goel. "Lease-Based Safety Protocol for Distributed System with Multiple Networks," U.S. Patent no. 6,775,703, August 2004.

8. Randal C. Burns and Darrell D. E. Long. "System and Method for Managing Authentication and Coherency in a Storage Area Network," U.S. Patent no. 6,792,424, September 2004.

9. Randal C. Burns, Darrell D. E. Long and Robert M. Rees. "System and Method for Restoring a File System from Backups in the Presence of Deletions," U.S. Patent no. 6,938,056, August 2005.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 114 of 411

10. Rajagopal Ananthanarayanan, Ralph Becker-Szendy, Randal C. Burns, Darrell D. E. Long, Jujjuri Venkateswararao, David M. Wolf and Jason C. Young. "Method, System and Computer Program Product for Implementing Copy-on-Write of a File," U.S. Patent no. 7,085,909, August 2006.

11. Jason C. Young, Rajagopal Ananthanarayanan, Randal C. Burns, Darrell D. E. Long, Robert M. Rees, Ralph A. Becker-Szendy, James J. Seeger and David M. Wolf. "Managing File System Versions," U.S. Patent no. 7,139,781, November 2006.

Professional Activities

Invited Lectures

2019 "Geomancy: Automated Performance Enhancement through Data Placement Optimization," IBM Research, Zürich, Organisation Européenne pour la Recherche Nucléaire (CERN), Sorbonne Université.

"Twizzler: An OS for Next-generation Memory Hierarchies," IBM Research, Zürich, Organisation Européenne pour la Recherche Nucléaire (CERN), Sorbonne Université.

2016 "RESAR: Reliable Storage at Exabyte Scale," IBM Research, Zürich, Université Paris–Descartes.

"Pilot: A Framework that Understands How to Do Performance Benchmarks the Right Way," Organisation Européenne pour la Recherche Nucléaire (CERN).

2014 "Horus: Fine-Grained Security for Large-Scale Storage," University of Houston.

"SNQoS: Scalable Shared-Nothing QoS for High-Performance Storage Systems," San Diego State University, University of New Orleans.

"Science and Cyber Security," Lockheed-Martin "Skunk Works" (Advanced Development Programs).

2013 "Horus: Fine-Grained Security for Large-Scale Storage," Laboratoire d'Informatique de Paris 6 (LIP6), Université Pierre et Marie Curie, Conservatoire National des Arts et Métiers, Université de Technologie en Sciences des Organisations et de la Décision de Paris–Dauphine.

"RESAR Storage: a System for Two-Failure Tolerant, Self-Adjusting Million Disk Storage Cluster," Conservatoire National des Arts et Métiers, Université Paris–Descartes, Université Pierre et Marie Curie (Laboratoire d'Informatique de Paris 6).

2012 "RESAR Storage: a System for Two-Failure Tolerant, Self-Adjusting Million Disk Storage Cluster," University of Oregon.

2011 "Science of Cyber Security," Sandia National Laboratories.

2010 "Scalable, Parallel Deduplication for Large Scale Data Systems," Pacific Northwest National Laboratory.

"Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup," Université de Technologie en Sciences des Organisations et de la Décision de Paris–Dauphine.

2009 "Maximizing Efficiency by Trading Storage for Computation," IBM-Amrita Cloud Symposium, Amrita University, Coimbatore, India.

"High-Performance Petascale File Systems," Université de Technologie en Sciences des Organisations et de la Décision de Paris–Dauphine.

"High-Performance Petascale File Systems," Commissariat à l'Energie Atomique, Direction des Applications Militaires et Direction des Ames Nucléaires.

2007 "Ceph: A Scalable, High-Performance Distributed File System," University of Technology, Sydney, Australia.

"Ceph: A Scalable, High-Performance Distributed File System," University of California, San Diego.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 115 of 411

2006 "Storage Management – Unchecked Challenges, Untapped Opportunities," Workshop on Distributed Data and Structures, Santa Clara University.

2005 "Storage Management – Unchecked Challenges, Untapped Opportunities," Storage Networking Summit, Bangalore, India.

"Ceph: A Distributed Object-based Storage System," Indian Institute of Science, Bangalore, India.

"Deep Store: Minimum Redundancy Archival Storage," Amrita University, Coimbatore, India.

2004 "Peta-scale Data Storage: Challenges and (a few) Solutions," Yahoo!

"Deep Store," University of Arizona.

"Deep Store," Vrije Universiteit.

2003 "Deep Store: Associative Archival Storage Architecture," Network Appliance.

2002 "Group-based Management of Distributed File Caches," Hewlett-Packard Laboratories.

"Group-based Management of Distributed File Caches," Technische Universität München.

2000 "Perspective on Academic Storage Research," IBM Research.

"Produce-Consumer Locking" and "Analytical Lease Modeling," San Francisco State University.

"The Storage Tank System," Université de technologie en sciences des organisations et de la décision de Paris–Dauphine

1997 "Differential Storage Management" and "Efficient Video-on-Demand Services," Université de technologie en sciences des organisations et de la décision de Paris–Dauphine

"Differential Storage Management," Queen Mary College, University of London.

"A Dynamic Disk Spin-down Technique for Mobile Computing," University of California, Riverside.

"Replication with Minimal Metadata," IBM Corporation.

1996 "Predicting File System Actions from Prior Events," University of California, San Diego.

1995 "Fault-Tolerance in Distributed Systems," WilTel Advanced Technology Group.

1994 "The REINAS System," University of California, Riverside.

1992 "A Study of the Reliability of Internet Sites," San Diego State University.

1991 "Swift – A Distributed Storage System for Multimedia," Matsushita Information Technology Laboratory.

"A Study of the Reliability of Internet Sites," INRIA (Institut National de Recherche en Informatique et Automatique).

"A Study of the Reliability of Internet Sites," University of California, San Diego.

"Swift – A Distributed Storage System for Multimedia," Hewlett-Packard Laboratories.

"Reliable Distributed Storage Systems," Sun Microsystems.

"A Study of the Reliability of Internet Sites," Bay Area Systems Seminar.

1990 "Swift: A Storage Architecture for Very Large Objects," Hewlett-Packard Laboratories.

"Estimating the Reliability of Regeneration-Based Replica Control Protocols," IBM Almaden Research and SRI International.

1989 "Availability and Reliability of Regeneration Protocols for Replicated Objects," Hewlett-Packard Laboratories.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 116 of 411

1988 "Optimistic Protocols for Replicated Data Management," Vanderbilt University and Stanford University.

1987 "The Management of Replication in a Distributed Computing System," Georgia Institute of Technology.

"Optimistic Algorithms for Replicated Data Management," Oxford University.

"The Management of Replication: An Overview of the Gemini Project," United States Naval Research Laboratory.

1986 "On Improving the Availability of Replicated Files," University of California, Berkeley.

Conference Organization

2024 General Chair: *Conference on Mass Storage Systems and Technologies* (MSST). Program Committee: *Usenix Technical Conference* (ATC).

2023 Program Committee: *Fourth Joint International Conference on Deep Learning, Big Data and Blockchain* (DBB).

2022 Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2021 Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2020 Program Chair: *Conference on Mass Storage Systems and Technologies* (MSST).

Program Committee: *Symposium on File and Storage Systems Technology* (FAST), *International Conference on Cloud Computing and Services Science* (CLOSER 2020).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2019 Program Committee: *IEEE International Conference on Machine Learning and Applications* (ICMLA 2019).

Reviewer: *International Conference on Cloud and Big Data Computing* (CBDC 2019).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2018 Program Committee: *IEEE Cluster Conference* (Cluster 2018).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2017 Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2016 Program Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2015 Program Committee: *Symposium on File and Storage Systems Technology* (FAST).

Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 117 of 411

2014 Program Committee: *International Conference on Advanced Communications and Computation* (INFOCOMP), *Conference on Mass Storage Systems and Technologies* (MSST).

General Chair: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

Reviewer: *International Symposium on Performance Analysis of Systems and Software* (ISPASS).

2013 Reviewer: *Electronics and Telecommunications Research Institute Journal* (ETRI), *IEEE Transactions on Neural Networks and Learning Systems*, *International Workshop on Combinatorial Algorithms*.

Program Committee: *Annual International Systems and Storage Conference* (SYSTOR), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Conference on Mass Storage Systems and Technologies* (MSST).

Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

Advisory Committee: *International Conference on Electronics, Computing and Communication Technologies* (CONECCT).

2012 Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

Program Committee: *Symposium on File and Storage Systems Technology* (FAST).

2011 Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

Program Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2010 Program Chair: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Petascale Data Storage Workshop* (PDSW), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

2009 Program Committee: *International Workshop on Software Support for Portable Storage* (IWSSPS), *Inaugural International Conference on Virtualization and Cloud Computing*, *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Petascale Data Storage Workshop* (PDSW).

Program Chair: *Web Information Systems Engineering* (WISE).

General Chair: *Symposium on Applications and the Internet* (SAINT).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

2008 Program Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Petascale Data Storage Workshop* (PDSW).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 118 of 411

2007 Program Committee: *Symposium on File and Storage Systems Technology* (FAST), *Petascale Data Storage Workshop* (PDSW).

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS), *Symposium on File and Storage Systems Technology* (FAST).

2006 Program Committee: *International Conference on Dependable Systems and Networks* (DSN), *Dependable Computing and Communications Symposium* (DCCS), *Symposium on Applications and the Internet* (SAINT).

Steering Committee: *Symposium on File and Storage Systems Technology* (FAST), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2005 Program Committee: *Symposium on File and Storage Systems Technology* (FAST), *Workshop on Storage Security and Survivability* (StorageSS), *First International Workshop on Software Support for Portable Storage* (IWSSPS).

2004 Program Committee: *Symposium on Applications and the Internet* (SAINT); *SIGMetrics*.

Steering Committee: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2003 Program Committee: *Usenix Technical Conference, Symposium on File and Storage Systems Technology* (FAST), *Very Large Data Bases* (VLDB), *International Conference on Distributed Computing Systems* (ICDCS), *IEEE International Performance, Computing, and Communications Conference* (IPCCC), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Symposium on File and Storage Systems Technology* (FAST), *Workshop on Mobile Computing Systems and Applications* (WMCSA), *IEEE International Performance, Computing, and Communications Conference* (IPCCC), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

2002 Program Committee: *IEEE International Performance, Computing, and Communications Conference* (IPCCC), *Workshop on Caching, Coherency and Consistency* (WC3), *Workshop on Distributed Data and Structures* (WDAS), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Symposium on File and Storage Systems Technology* (FAST), *Workshop on Mobile Computing Systems and Applications* (WMCSA), *IEEE International Performance, Computing, and Communications Conference* (IPCCC).

2001 Founding Program Chair: *Symposium on File and Storage Systems Technology* (FAST).

Program Chair: *IEEE International Performance, Computing, and Communications Conference* (IPCCC).

Program Committee: *International Conference on Distributed Computing Systems* (ICDCS).

2000 Program Committee: *Communication Networks and Distributed Systems Modeling and Simulation Conference* (CNDS), *Research Issues in Data Engineering* (RIDE), *Hot Interconnects, Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Workshop on Mobile Computing Systems and Applications* (WMCSA), *Symposium on Applications and the Internet* (SAINT).

1999 General Chair: *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Program Committee: *Communication Networks and Distributed Systems Modeling and Simulation Conference* (CNDS), *IEEE International Performance, Computing, and Communications Conference* (IPCCC).

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 119 of 411

Steering Committee: *Symposium on Applications and the Internet* (SAINT).

1998 Program Committee: *Symposium on Reliable Distributed Systems* (SRDS), *Fault Tolerant Computing Systems* (FTCS), *Usenix Technical Conference.*

Steering Committee: *IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA).

Advisory Committee: *Hawaii International Conference on System Sciences* (HICSS).

1997 General Chair: *IEEE Computer Society CompCon.*

Program Committee: *International Conference on Parallel and Distributed Systems* (ICPDS), *Communications Networks and Distributed Systems Modeling and Simulation Conference* (CNDS), *Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Hot Topics in Operating Systems VI* (HotOS).

1996 General Chair: *IEEE Computer Society/National Science Foundation Workshop on Workspaces in the Information Age.*

1996 Program Chair: *IEEE Computer Society CompCon.*

Program Committee: *International Conference on Distributed Computing Systems* (ICDCS), *Annual Simulation Symposium, International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

Steering Committee: *Hot Topics in Operating Systems V* (HotOS).

1995 Program Committee: *Symposium on Reliable Distributed Systems* (SRDS), *Usenix Technical Conference, Services in Distributed and Networked Environments* (SDNE), *International Conference on Distributed Computing Systems* (ICDCS), *Annual Simulation Symposium, International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

1994 Founding General Chair: *Workshop on Mobile Computing Systems and Applications* (WMCSA).

1994 Program Committee: *Applied Data Bases; First International Workshop on Services in Distributed and Networked Environments, International Conference on Distributed Computing Systems* (ICDCS), *Annual Simulation Symposium, International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

1993 Program Committee: *International Conference on Parallel and Distributed Systems* (ICPDS), *IEEE Workshop on Advances in Parallel and Distributed Systems, International Conference on Distributed Computing Systems* (ICDCS), *Annual Simulation Symposium, International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

1992 Program Committee: *Annual Simulation Symposium, Second Workshop on the Management of Replicated Data* (WMRD), *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (MASCOTS).

1991 Local Arrangements Chair and Program Committee: *Second Workshop on the Management of Replicated Data* (WMRD).

1990 Publicity Chair: *First Workshop on the Management of Replicated Data* (WMRD).

1989 Program Committee: Usenix Association/ACM/IEEE-CS, *Workshop on Experiences with Building Distributed and Multiprocessor Systems.*

1988 Program Committee: IFIP Working Group 6.5, *International Working Conference on Message Handling Systems and Distributed Applications.*

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 120 of 411

Membership in Professional Associations

Year	Rank	Professional Society
2019	Member	International Association for Cryptologic Research (IACR)
2012	Member	Armed Forces Communications & Electronics Association
2008	Fellow	American Association for the Advancement of Science (AAAS)
2006	Fellow	Institute of Electrical & Electronics Engineers (IEEE)
2006	Member	American Society for Engineering Education (ASEE)
1991	Member	Usenix Association
1989	Member	Sigma Xi (Scientific Research Society)
1987	Member	IEEE Computer Society
1981	Member	Association for Computing Machinery (ACM)

Consulting

Years	Organization	Role
2015–	Wilson, Sonsini, Goodrich & Rosati	Technical Expert
	Fish & Richardson	Technical Expert
2015	Foley & Lardner	Technical Expert
	DLA Piper	Technical Expert
2014–	Latham & Watkins	Technical Expert
2014	Beck & Hall	Technical Expert
2013–15	Faegre, Baker & Daniels	Technical Expert
	Wilmer Cutler Pickering Hale	Technical Expert
2013–14	County of Tehama	Technical Expert
2012–13	Klarquist Sparkman	Technical Expert
	Wilson, Sonsini, Goodrich & Rosati	Technical Expert
2011–14	Morgan, Lewis & Bockius	Technical Expert
	Fish & Richardson	Technical Expert
2011–13	Connoly Bove	Technical Expert
2011–12	Haynes & Boone	Technical Expert
2011	Fujitsu Semiconductor America	Consultant
2010–12	Quinn Emanuel Urquhart & Sullivan	Technical Expert
	Fish & Richardson	Technical Expert
	Simpson, Thacher & Bartlett	Technical Expert
2009	Preston, Gates & Ellis	Technical Expert
2008–09	Weil, Gotshal & Mange	Technical Expert
2008–10	Hewlett-Packard Laboratories	Consultant
2007–	Center for Communications Research	Adjunct
2006–10	Weil, Gotshal & Manges	Technical Expert
2006–14	Fish & Richardson	Technical Expert
2004–05	Faegre & Benson	Technical Expert
2004	Siemens Technology-to-Business Center	Consultant
2002–	MITRE Corporation	JASON
2002–05	Fish & Richardson	Technical Expert
2001–03	Hecker Law Group	Technical Expert
	Bartlit, Beck, Herman, Palenchar & Scott	Technical Expert
	Jenner & Block	Technical Expert
2000	Sony Music Ventures	Consultant
1997–02	Chief of Naval Research	Consultant

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 121 of 411

1996	Chatham Group	Technical Expert
1995–11	IBM Almaden Research Center	Visiting Scientist

Editorial Duties

2023 Associate Editor, *Springer Journal of Big Data (JOBD)*

2017–20 Editor-in-Chief, *Letters of the Computer Society.*

2016–19 Associate Editor, *IEEE Transactions on Computers.*

2011 Guest Editor, *Information Systems Databases: Creation, Management and Utilization,* Elsevier, vol. 36, no. 3, May 2011.

2010–16 Editor-in-Chief, *ACM Transactions on Storage.*

2009 Guest Editor, *Web Information Systems Engineering* (WISE 2009), LNCS 5802.

2004–10 Associate Editor, *ACM Transactions on Storage.*

2002–03 Editorial Board, *Mobile Computing and Communications Review.*

1993–97 Editorial Board, *International Journal in Computer Simulation.*

1990–93 Editor, *Bulletin of the IEEE Computer Society Technical Committee on Operating Systems and Applications Environments.*

1990–93 Associate Editor, *International Journal in Computer Simulation.*

Reviewer of Technical Papers and Proposals

2023 ACM Transactions on Computer Systems.

2022 ACM Transactions on Computer Systems; The Journal of Super Computing; Performance Evaluation.

2021 Transactions on Dependable and Secure Computing.

2017 Information Security Journal: A Global Perspective; ACM Transactions on Storage.

2016 ACM Transactions on Modeling and Performance Evaluation of Computing Systems; Israel Science Foundation; Flanders Innovation & Entrepreneurship.

2012 ACM Transactions on Storage; Journal of Computer Science and Technology; IEEE Transactions on Neural Networks and Learning Systems; IEEE Transactions of Reliability.

2009 ACM Transactions on Storage; The Journal of Systems and Software.

2008 ACM Transactions on Storage; IEEE Transactions on Computers; International Conference on Networking, Architecture, and Storage.

2007 ACM Transactions on Storage; IEEE Transactions on Computers; Performance and Dependability Symposium (PDS).

2006 IEEE Transactions on Computers; ACM Transactions on Storage; IEEE Transactions on Magnetics; International Symposium on Stabilization, Safety, and Security of Distributed Systems; International Parallel and Distributed Processing Symposium; University of California, MICRO Program; International Conference on Dependable Systems and Networks.

2005 ACM Transactions on Storage; IEEE/ACM Transactions on Networking.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 122 of 411

2004 ACM Transactions on Storage; IEEE Transactions on Computers; IEEE Transactions on Parallel and Distributed Systems; International Journal of Computers and Applications.

2003 Symposium on Operating Systems Principles; IEEE Transactions on Broadcasting; Computer Journal; IEEE Communications Letters.

2002 Transactions on Knowledge and Data Engineering; Transactions on Broadcasting; International Conference on Parallel Processing; IEEE Communications Letters; Symposium on Operating System Design and Implementation; Western Multiconference.

2001 IEEE Personal Communications; Workshop on Hot Topics in Operating Systems (HotOS); ETRI Journal; International Conference on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems; International Conference on Distributed Computing Systems; Symposium on Operating Systems Principles; Western Multiconference.

2000 Transactions on Knowledge and Data Engineering; Symposium on Reliable Distributed Systems; Communications of the ACM; IEEE Personal Communications; Transactions on Computer Systems; Transactions on Modeling and Computer Simulation; Workshop on Wireless Networks and Mobile Computing.

1999 ACM/Baltzer Mobile Networks and Applications; IEEE Software; IEEE Transactions on Software Engineering; International Conference on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems; Usenix Technical Conference; Journal of Parallel and Distributed Computing.

1998 IEEE Software; IEEE Transactions on Parallel and Distributed Systems; ACM Transactions on Computer Systems; Conference on Computing and Telecommunications; International Conference on High Performance Computing; International Conference on Parallel Processing.

1997 National Science Foundation; IEEE Computer; IEEE Transactions on Parallel and Distributed Systems; Symposium on Reliable Distributed Systems; Software – Practice & Experience; IEEE Software; Computer Journal; Transactions on Knowledge and Data Engineering.

1996 National Science Foundation; Computer Journal; University of California, MICRO program; Information Systems Journal; IEEE Transactions on Parallel and Distributed Systems.

1995 University of California, MICRO program; National Science Foundation; Computer Journal; Distributed Systems Engineering Journal; Computing Surveys; Dependable Computing and Critical Applications Conference; IEEE Transactions on Knowledge and Data Engineering.

1994 IEEE Computer; High Performance Distributed Computing; California Space Institute; Operating Systems Design and Implementation; International Journal in Computer Simulation; Information Processing Letters; Journal of Distributed and Parallel Data Bases; IEEE Software. ACM Transactions on Computer Systems.

1993 International Conference on Distributed Computing Systems; Distributed and Parallel Databases; Hawaii International Conference on Systems Sciences; International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems; International Symposium on Autonomous Decentralized Systems; IEEE Transactions on Parallel Distributed Systems; IEEE Transactions on Computers; Information Processing Letters; IEEE Computer.

1992 International Symposium on Autonomous Decentralized Systems; IEEE Transactions on Parallel Distributed Systems; IEEE/ACM Transactions on Networking; Information Systems; IFIP Dependable Computing in Critical Applications; IEEE Transactions on Software Engineering; IEEE Transactions on Computers; International Conference on Data Engineering; IEEE Transactions on Knowledge and Data Engineering; University of California, MICRO Program; IEEE Computer; Information Processing Letters.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 123 of 411

1991 International Conference on Data Engineering; International Journal on Very Large Databases; ACM Conference on Principles of Data Base Systems; IEEE Transactions on Parallel and Distributed Systems; IEEE Transactions on Knowledge and Data Engineering; Computing Systems; University of California, MICRO Program; IEEE Computer; ACM Transactions on Computer Systems; Information Processing Letters.

1990 IEEE Transactions on Parallel and Distributed Systems; IEEE Transactions on Knowledge and Data Engineering; Computing Systems; Communications of the ACM; 1991 SIGMETRICS Conference on Measurement and Modeling of Computer Systems; University of California, MICRO Program; National Science Foundation; IEEE Computer; ACM Transactions on Computer Systems; Information Processing Letters; IEEE Software.

1989 National Science Foundation; IEEE Computer; ACM Transactions on Computer Systems; Information Processing Letters; IEEE Software; International Conference on Data Engineering; International Conference on Distributed Computing Systems.

1988 International Conference on Distributed Computing Systems.

1987 International Conference on Data Engineering.

Reviewer for Publishers

2011 John Wiley & Sons.

2003 Prentice-Hall.

2002 John Wiley & Sons.

1997 Academic Press; Addison-Wesley; Gordon and Beach.

1995 Addison-Wesley; Benjamin/Cummings; Springer-Verlag.

1994 Benjamin/Cummings; Morgan-Kaufman.

1993 Kluwer Academic Publishers; Norton; Benjamin/Cummings; McGraw-Hill; John Wiley & Sons.

1991–93 Aksen Associates.

1988–91 Addison-Wesley.

Government Service

2020 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Nuclear Security Administration. Department of Defense.

2019 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Nuclear Security Administration. Department of Defense. Department of State.

2018 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Nuclear Security Administration. Department of Defense.

2017 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Nuclear Security Administration. Department of Defense.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 124 of 411

2016 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Security Directorate Advisory Committee (Pacific Northwest National Laboratory). Computer Information Science External Review Board (Sandia National Laboratory). National Nuclear Security Administration. Department of Energy. Department of Defense.

2015 Intelligence Science and Technology Experts Group (ISTEG) (National Academies of Sciences, Engineering and Medicine committee). National Security Directorate Advisory Committee (Pacific Northwest National Laboratory). Computer Information Science External Review Board (Sandia National Laboratory). National Nuclear Security Administration. Department of Energy. Department of Defense.

2014 National Security Directorate Advisory Committee (Pacific Northwest National Laboratory). Computer Information Science External Review Board (Sandia National Laboratory). National Nuclear Security Administration. Department of Energy. Department of Defense.

2013 National Research Council, Committee on Science and Technology for Defense Warning. National Nuclear Security Administration. Department of Energy. Department of Defense.

2012 National Research Council, Committee for Science and Technology Challenges to U.S. National Security Interests. National Research Council, Committee on Science and Technology for Defense Warning, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). National Nuclear Security Administration. Department of Energy. Department of Defense.

2011 National Research Council, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). National Research Council, report reviewer. National Nuclear Security Administration. Department of Defense. Department of the Navy.

2010 Department of Energy. National Nuclear Security Administration. Department of Defense. National Research Council, report reviewer.

2009 United States Army Laboratory Assessment Group. Department of Energy. National Nuclear Security Administration. Department of Defense. Department of Energy, Office of Science (Early Career panel).

2008 United States Army Laboratory Assessment Group. National Research Council, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). Department of Energy. National Nuclear Security Administration. Department of Defense.

2007 External Review Committee, United States Naval Postgraduate School. United States Army Technology Objectives Review Panel. National Science Foundation (review panels). National Research Council, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). National Research Council, Reviewer. Department of Energy. National Nuclear Security Administration. Department of Defense.

2006 National Science Foundation (review panels). National Research Council, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). National Research Council, Reviewer. Department of Energy. National Nuclear Security Administration. Department of Defense.

2005 National Research Council, Standing Committee on Technology Insight-Gauge, Evaluate, and Review (TIGER). Department of Energy. National Nuclear Security Administration. Department of Defense.

2004 National Research Council, Defense Intelligence Agency Technology Forecast and Reviews Committee. National Nuclear Security Administration. Department of Defense. Los Alamos National Laboratory, hiring committee for Associate Director for Nuclear Weapons Physics and Principal Associate Director for Nuclear Weapons Programs.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 125 of 411

2003 National Science Foundation (two Information Technology Research panels). Department of Energy. National Nuclear Security Administration. Department of Defense.

2002 National Science Foundation (two Information Technology Research panels). Department of Energy. National Nuclear Security Administration.

2001 National Science Foundation. Department of Energy. National Nuclear Security Administration.

2000 National Science Foundation (Information Technology Research panel, National Environmental Observatory Network planning panel, CAREER panel).

1999 National Science Foundation (site review of Data Storage Systems Center, panel on Engineering Research Centers). Department of Defense.

1998 National Science Foundation (site review of Data Storage Systems Center). Department of the Navy.

1997 Department of the Navy.

Service to Professional Societies

2020 IEEE Fellows Evaluation Committee.

2014 IEEE Fellows Evaluation Committee.

2013 IEEE Fellows Evaluation Committee.

2011–12 Member and Chair Emeritus, IEEE Reynold B. Johnson Information Storage Systems Award Committee.

2011 IEEE Fellows Evaluation Committee, IEEE Fellow Society/Technical Council Evaluator.

2010 IEEE Fellows Evaluation Committee.

2008–11 Chair, IEEE Reynold B. Johnson Information Storage Systems Award Committee, IEEE Technical Field Awards Council.

2007 IEEE Fellows Evaluation Committee.

2006–08 IEEE Reynold B. Johnson Information Storage Systems Award Committee.

2000–02 Chair, Usenix Association Scholarship and Research Committee.

1998–02 Usenix Association Scholarship and Research Committee.

1997– University Liaison for the Usenix Association.

1997 Committee on the Future of the Usenix Association.

1996– Executive Committee, *IEEE Technical Committee on Operating Systems.*

1995 IEEE/ACM focus group on Software Engineering Body of Knowledge and Recommended Practices.

1993–96 Chair, *IEEE Technical Committee on Operating Systems and Applications Environments.*

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 126 of 411

University Service

Senate Service

2018–19 Committee on Research.

2016–17 Committee on Academic Freedom.

2008–09 Committee on Academic Personnel *ad hoc* subcommittee.

2007–08 Committee on Academic Personnel *ad hoc* subcommittee.

2006–07 Committee on Research.

2005–06 Committee on Research; Committee on Academic Personnel *ad hoc* subcommittee.

2003–04 Committee on Committees; Committee on Academic Personnel *ad hoc* subcommittee.

2001–02 Committee on Academic Personnel *ad hoc* subcommittee; Committee on Research.

2000–01 Committee on Research; Chair, Committee on Academic Personnel *ad hoc* subcommittee;

2000–01 Silicon Valley Regional Center Task Force; California Institutes for Science and Innovation proposal (with P. Mantey & Berkeley colleagues).

1999–00 Committee on Academic Personnel *ad hoc* subcommittee; Computing and Telecommunications Committee.

1998–99 Computing and Telecommunications Committee.

1995–97 Senate Rules Committee, Information Technology Disaster Preparedness Task Force.

Campus Service

2015– Genomics Institute Steering Committee.

2004–05 Confidential personnel review for senior administrator.

2000–04 Chair, Engineering II Building Committee.

1998–99 Confidential personnel review for senior administrator.

1998 Faculty mentor, California Alliance for Minority Participation (CAMP).

1997 Faculty mentor, California Alliance for Minority Participation (CAMP).

1996–97 Information Technology Disaster Preparedness Task Force.

1994–95 Task Force on Computer and Network Security, Summer Science Honors Institute.

1990–91 Technical Consultant to Computing and Telecommunication Services (CATS) for University computing facilities.

1988–89 Applied Sciences Building Remodeling Committee. Open Access Computing Committee.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 127 of 411

School of Engineering Service

2017– Strategic Faculty Advisor to the School of Engineering.

2004–10 Associate Dean for Research and Graduate Studies.

1998–01 Associate Dean for Research.

1995–96 Electrical Engineering/Applied and Engineering Mathematics Committee.

1988–90 Natural Sciences Division Space Committee.

Departmental Service

2015–16 Personnel Review Committee; Chair, *ad hoc* Personnel Committee; Security & Privacy faculty hiring committee.

2014–15 Personnel Review Committee; Chair, *ad hoc* Personnel Committee; Experimental Computer Science faculty hiring committee.

2013–14 Department Chair Selection Committee; Personnel Review Committee; Data Sciences faculty hiring committee.

2011–12 Personnel Best Practices Committee; Personnel Review Committee; Chair, *ad hoc* Personnel Committee.

2010–11 Department Awards Coordinator; Chair, *ad hoc* Personnel Committee; Chair, Adjunct Committee.

2009–10 Director for Research, Computer Science Department; Chair, Adjunct Committee.

2008–09 Director for Research, Computer Science Department; Chair, Adjunct Committee.

2007–08 Director for Research, Computer Science Department; Chair, Adjunct Committee; Seminar Coordinator, Computer Science Department.

2006–07 Personnel Review Committee; Director for Research, Computer Science Department.

2005–06 Personnel Review Committee; Director for Research, Computer Science Department; Department of Information Systems and Technology Management.

2004–05 Director for Research, Computer Science Department; Master of Ceremonies, Engineering 2 dedication; Information Technologies Institute (ITI) Executive Committee.

2003–04 Director for Research, Computer Science Department; Information Technologies Institute (ITI) Executive Committee; School of Engineering Technical Services Committee.

2002–03 Chair, Promotion Review Subcommittee; Computer Science Faculty Recruitment Committee; ISTM Chair Recruitment; Faculty Recruitment Committee.

2001–02 Chair, Promotion Review Subcommittee; Computer Science Faculty Recruitment Committee; Computer Science Department Facilities Director; Computing Infrastructure Committee; Faculty Recruitment Committee.

2000–01 Computer Science Faculty Recruitment Committee; Computer Science Department Facilities Director; Computing Infrastructure Committee; Faculty Recruitment Committee.

1999–00 Computer Science Faculty Recruitment Committee; Computer Science Department Facilities Director; Computing Infrastructure Committee; Faculty Recruitment Committee.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 128 of 411

1998–99 Computer Science Faculty Recruitment Committee; Computer Science Department Facilities Director; Computing Infrastructure Committee; Faculty Recruitment (Software Engineering); Information Systems Management Major.

1997–98 Computer Science Faculty Recruitment Committee; Computer Science Department Facilities Director; Information Systems Management Major.

1991–96 Computer Science Facilities Director.

1990–91 Computer & Information Sciences Assistant Facilities Director.

1989–90 Computer & Information Sciences Department Faculty Recruitment Committee, Co-Chair for the Operating Systems position.

1989 Computer & Information Sciences Department Representative to Electronics Engineering Program Development Committee.

1988–89 Research Funding Director, Computer & Information Sciences Department (responsible for the CIS/CE 1989 grant proposal to the National Science Foundation Institutional Infrastructure Program).

1988–89 Computer & Information Sciences Department Representative on Computer Engineering Department Recruitment Committee.

System-wide Service

2020–23 Academic Council's Special Committee on Laboratory Issues (ACSCOLI).

2011–16 Academic Council's Special Committee on Laboratory Issues (ACSCOLI).

2007–13 Science & Technology Committee for both Lawrence Livermore and Los Alamos National Laboratories (University of California, Office of the President).

2007–14 Multiple Directorate and Capability review committees for Lawrence Livermore and Los Alamos National Laboratories (University of California, Office of the President).

2007–08 Advisory Board, Department of Computer Engineering, Santa Clara University; Predictive Knowledge Systems Advisory Committee, Lawrence Livermore National Laboratory (University of California, Office of the President).

2007 Nonproliferation, Homeland and International Security, Directorate Review Committee, Lawrence Livermore National Laboratory; Petascale Computing Management Review Committee, Los Alamos National Laboratory (University of California, Office of the President).

2005–20 Faculty Advisory Group for the South Asia Initiative.

2005– Executive Committee, Center for Information technology Research in the Interest of Society (CITRIS).

2003–07 National Security Panel and Intelligence Panel for the National Laboratories (University of California, Office of the President).

2002–03 Chair, University-wide Committee on Research Policy; National Security Panel, Science & Technology Panel, and President's Council (University-wide Senate Representative).

2001–02 Vice Chair, University-wide Committee on Research Policy; Screening Committee for the Director of Lawrence Livermore National Laboratory; National Security Panel, Science & Technology Panel, President's Council (University-wide Senate Representative).

2000–03 University-wide Committee on Research Policy.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 129 of 411

TEACHING 1988–89

				Enrolled	%Eval Retd	Shared?
Fall	CIS	104	Introduction to Compiler Design	43	74	no
	CIS	301	Supervised Teaching Experience	1	n/a	no
Winter	CIS	221	Advanced Operating Systems	5	100	no
Spring	CIS	104	Introduction to Compiler Design	31	74	no
	CIS	301	Supervised Teaching Experience	1	100	no

TEACHING 1989–90

				Enrolled	%Eval Retd	Shared?
Fall	CIS	204	Compiler Design	18	94	no
	CIS	297	Independent Study/Research	1	n/a	no
	CIS	299	Thesis Research	1	n/a	no
Winter	CIS	104	Introduction to Compiler Design	17	88	no
	CIS	195	Independent Study/Research	1	n/a	no
	CIS	221	Advanced Operating Systems	8	88	no
	CIS	299	Thesis Research	3	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no
Spring	CIS	104	Introduction to Compiler Design	22	86	no
	CIS	297	Independent Study/Research	2	n/a	no
	CIS	299	Thesis Research	3	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no

TEACHING 1990–91

				Enrolled	%Eval Retd	Shared?
Fall	CIS	204	Compiler Design	10	100	no
	CIS	221	Advanced Operating Systems	12	75	no
	CIS	299	Thesis Research	3	n/a	no
Winter	CIS	104A	Fundamentals of Compiler Design I	18	67	no
	CIS	198	Independent Study/Research	2	n/a	no
	CIS	297	Independent Study/Research	1	n/a	no
	CIS	299	Thesis Research	3	n/a	no
	CIS	301	Supervised Teaching Experience	1	n/a	no
Spring	CIS	104B	Fundamentals of Compiler Design II	17	76	no
	CIS	195	Independent Study/Research	2	n/a	no
	CIS	297	Independent Study/Research	2	n/a	no

TEACHING 1991–92

				Enrolled	%Eval Retd	Shared?
Fall	CIS	221	Advanced Operating Systems	9	100	no
	CIS	198	Independent Study/Research	1	n/a	no
	CIS	297	Independent Study/Research	2	n/a	no
	CIS	299	Thesis Research	1	n/a	no
Winter	CIS	198	Independent Study/Research	2	n/a	no
	CIS	297	Independent Study/Research	3	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 130 of 411

	CIS	299	Thesis Research	3	n/a	no
Spring	CIS	104B	Fundamentals of Compiler Design II	14	79	no
	CIS	195	Independent Study/Research	2	n/a	no
	CIS	198	Independent Study/Research	1	n/a	no
	CIS	203	Programming Languages & Environments	8	88	no
	CIS	297	Independent Study/Research	3	n/a	no
	CIS	299	Thesis Research	5	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no

TEACHING 1992–93

				Enrolled	%Eval Retd	Shared?
Fall	CIS	111	Introduction to Operating Systems	43	80	no
	CIS	299	Thesis Research	6	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no
Winter	CIS	198	Independent Study/Research	1	n/a	no
	CIS	204	Compiler Design	5	100	no
	CIS	297	Independent Study/Research	4	n/a	no
	CIS	299	Thesis Research	3	n/a	no
Spring	CIS	195	Senior Thesis Research	1	n/a	no
	CIS	221	Advanced Operating Systems	8	100	no
	CIS	297	Independent Study/Research	4	n/a	no
	CIS	299	Thesis Research	3	n/a	no
	Stev	003	Privacy & Computer Security	11	73	no

TEACHING 1993–94

				Enrolled	%Eval Retd	Shared?
Fall	CIS	111	Introduction to Operating Systems	36	78	no
	CIS	297	Independent Study/Research	4	n/a	no
	CIS	299	Thesis Research	3	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no
Winter	CIS	204	Compiler Design	5+1	100	no
	CIS	297	Independent Study/Research	10	n/a	no
	CIS	299	Thesis Research	1	n/a	no
Spring	CIS	203	Programming Languages & Environments	21	85	no
	CIS	297	Independent Study/Research	8	n/a	no
	CIS	299	Thesis Research	5	n/a	no

TEACHING 1994–95

				Enrolled	%Eval Retd	Shared?
Fall	CIS	111	Introduction to Operating Systems	56	68	no
	CIS	195	Senior Thesis Research	1	n/a	no
	CIS	221	Advanced Operating Systems	15	60	no
	CIS	297	Independent Study/Research	3	n/a	no
	CIS	299	Thesis Research	9	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no
Winter	CIS	198	Independent Study/Research	3	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 131 of 411

	CIS	297	Independent Study/Research	5	n/a	no
	CIS	299	Thesis Research	10	n/a	no
Spring	CIS	195	Senior Thesis Research	1	n/a	no
	CIS	203	Programming Languages & Environments	16+3	95	no
	CIS	297	Independent Study/Research	2	n/a	no
	CIS	299	Thesis Research	10	n/a	no
	CIS	301	Supervised Teaching Experience	1	100	no

TEACHING 1995–96

				Enrolled	%Eval Retd	Shared?
Fall	CIS	111	Introduction to Operating Systems	18	78	no
	CIS	204	Compiler Design	6	83	no
	CIS	297	Independent Study/Research	3	n/a	no
	CMPE	297	Independent Study/Research	1	n/a	no
	CIS	299	Thesis Research	3	n/a	no
Winter	CIS	198	Independent Study/Research	2	n/a	no
	CIS	297	Independent Study/Research	4	n/a	no
	CIS	299	Thesis Research	3	n/a	no
Spring	CIS	195	Senior Thesis Research	1	n/a	no
	CIS	221	Advanced Operating Systems	13	77	no
	CIS	297	Independent Study/Research	3	n/a	no
	CMPE	297	Independent Study/Research	1	n/a	no
	CIS	299	Thesis Research	6	n/a	no
	CMPE	299	Thesis Research	1	n/a	no

TEACHING 1996–97

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	111	Introduction to Operating Systems	20	70	no
	CMPS	195	Thesis Research	1	n/a	no
	CMPS	297	Independent Study/Research	8	n/a	no
	CMPE	297	Independent Study/Research	4	n/a	no
	CMPS	299	Thesis Research	5	n/a	no
	CMPE	299	Thesis Research	3	n/a	no
	CMPS	301	Supervised Teaching Experience	1	100	no
Winter	CMPS	290X	Cryptography	6	67	no
	CMPS	297	Independent Study/Research	5	n/a	no
	CMPS	299	Thesis Research	6	n/a	no
	CMPE	299	Thesis Research	2	n/a	no
Spring	CMPS	090X	Internet Security & Privacy	19	58	no
	CMPS	221	Advanced Operating Systems	8	88	no
	CMPS	297	Independent Study/Research	6	n/a	no
	CMPS	299	Thesis Research	6	n/a	no
	CMPE	299	Thesis Research	3	n/a	no

TEACHING 1997–98

				Enrolled	%Eval Retd	Shared?

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 132 of 411

Fall	CMPS		Sabbatical Leave			
	CMPS	297	Independent Study/Research	1	n/a	no
	CMPS	299	Thesis Research	9	n/a	no
	CMPE	299	Thesis Research	3	n/a	no
Winter	CMPS	112	Comparative Programming Languages	58	60	no
	CMPS	297	Independent Study/Research	3	n/a	no
	CMPS	299	Thesis Research	10	n/a	no
	CMPE	299	Thesis Research	2	n/a	no
	CMPS	301	Supervised Teaching Experience	1	n/a	no
Spring	CMPS	221	Advanced Operating Systems	9	100	no
	CMPS	297	Independent Study/Research	2	n/a	no
	CMPS	299	Thesis Research	8	n/a	no
	CMPE	299	Thesis Research	1	n/a	no

TEACHING 1998–99

				Enrolled	%Eval Retd	Shared?
Fall	CMPE	299	Thesis Research	1	n/a	no
	CMPS	297	Independent Study/Research	3	n/a	no
	CMPS	299	Thesis Research	6	n/a	no
Winter	CMPE	299	Thesis Research	1	n/a	no
	CMPS	297	Independent Study/Research	4	n/a	no
	CMPS	299A	Thesis Research	6	n/a	no
Spring	CMPE	299	Thesis Research	1	n/a	no
	CMPS	111	Introduction to Operating Systems	62	76	no
	CMPS	221	Advanced Operating Systems	6	100	no
	CMPS	297	Independent Study/Research	2	n/a	no
	CMPS	299	Thesis Research	5	n/a	no
	CMPS	301	Supervised Teaching Experience	1	100	no
Summer	CMPS	198	Independent Study/Research	1	n/a	no

TEACHING 1999–00

				Enrolled	%Eval Retd	Shared?
Fall	CMPE	299	Thesis Research	1	n/a	no
	CMPS	111	Introduction to Operating Systems	45	63	no
	CMPS	297	Independent Study/Research	2	n/a	no
	CMPS	299	Thesis Research	6	n/a	no
	CMPS	301	Supervised Teaching Experience	1	100	no
Winter	CMPS	112	Comparative Programming Languages	47	73	no
	CMPS	290X	Cryptography	3	0	no
	CMPS	297	Independent Study/Research	3	n/a	no
	CMPS	299	Thesis Research	2	n/a	no
Spring	CMPS	198	Independent Study/Research	1	n/a	no
	CMPS	297	Independent Study/Research	2	n/a	no
	CMPS	299	Thesis Research	3	n/a	no

TEACHING 2000–01

				Enrolled	%Eval Retd	Shared?

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 133 of 411

Fall	CMPS	299C	Thesis Research	2	n/a	no
	CMPS	301	Supervised Teaching Experience	1	n/a	no
	ISM	150	Introduction To Information Networks	36	0	no
Winter	CMPS	112	Comparative Programming Languages	75	77	no
	CMPS	297A	Individual Study/ Research	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
	CMPS	301	Supervised Teaching Experience	1	n/a	no
Spring	CMPS	297A	Individual Study/ Research	3	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Summer	CMPS	297A	Individual Study/ Research	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no

TEACHING 2001–02

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	198	Individual Study/Research	2	n/a	no
	CMPS	297A	Individual Study/Research	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
	ISM	150	Introduction To Information Networks	54	69	no
Winter	CMPS	195	Senior Thesis Research	1	n/a	no
	CMPS	198	Individual Study/Research	1	n/a	no
	CMPS	221	Advanced Operating Systems	17	82	no
	CMPS	221-50	Advanced Operating Systems	9	89	no
	CMPS	297A	Individual Study/Research	2	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Spring	CMPS	112	Comparative Programming Languages	62	61	no
	CMPS	198	Individual Study/Research	1	n/a	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study/Research	1	n/a	no
	CMPS	297B	Individual Study/Research	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Summer	CMPS	297A	Individual Study/Research	1	n/a	no

TEACHING 2002–03

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	111	Introduction To Operating Systems	53	92	no
	CMPS	280S	Seminar On Computer Systems	10	n/a	no
	CMPS	297A	Individual Study/Research	2	n/a	no
	CMPS	297B	Individual Study/Research	1	n/a	no
Winter	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study/Research	3	n/a	no
	CMPS	297B	Individual Study/Research	1	n/a	no
Spring	CMPE	299B	Thesis Research	1	n/a	no
	CMPS	229	Storage Systems	8	100	no
	CMPS	299B	Thesis Research	1	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 134 of 411

TEACHING 2003–04

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	13H	Introduction Programming/Data Structures	9	77	no
	CMPS	280S	Seminar Computer Systems	14	0	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
Winter	CMPS	111	Introduction Operating Systems	47	70	no
	CMPS	297A	Individual Study	3	n/a	no
	CMPS	299B	Thesis Research	2	n/a	no
Spring	CMPE	297B	Individual Study	1	n/a	no
	CMPS	290S	Adv Topics: Computer Systems	7	100	no
	CMPS	297A	Individual Study	2	n/a	no
	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no

TEACHING 2004–05

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
Winter	CMPE	299A	Thesis Research	1	n/a	no
	CMPE	299B	Thesis Research	1	n/a	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	280S	Seminar Computer Systems	18	50	no
Spring	CMPS	111	Introduction to Operating Systems	58	76	no
	CMPS	297A	Individual Study	2	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
Summer	CMPS	297A	Individual Study	1	n/a	no

TEACHING 2005–06

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	2	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299B	Thesis Research	2	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no
Winter	CMPS	60N	Beginning Programming	3	133	no
	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	299B	Thesis Research	3	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no
Spring	CMPS	111	Introduction to Operating Systems	24	108	yes

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 135 of 411

	CMPS	280	Seminar on Computer Systems	13	8	no
	CMPS	299A	Thesis Research	3	n/a	no
	CMPS	299B	Thesis Research	2	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no

TEACHING 2006–07

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	299B	Thesis Research	2	n/a	no
	CMPS	299B	Thesis Research	2	n/a	no
Winter	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	299B	Thesis Research	2	n/a	no
Spring	CMPS	111	Introduction to Operating Systems	14	86	no
	CMPS	198	Individual Study or Research	1	n/a	no
	CMPS	280S	Seminar on Computer Systems	16	13	no
	CMPS	299B	Thesis Research	2	n/a	no

TEACHING 2007–08

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	221	Advanced Operating Systems	10	90	no
	CMPS	280S	Seminar on Computer Systems	10	50	no
	CMPS	299B	Thesis Research	2	n/a	no
Winter	CMPS	297A	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no
Spring	CMPS	111	Introduction to Operating Systems	38	71	no
	CMPS	198	Individual Study or Research	1	n/a	no
	CMPS	280S	Seminar on Computer Systems	13	62	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no

TEACHING 2008–09

				Enrolled	%Eval Retd	Shared?
Fall	CMPE	297B	Individual Study	1	n/a	no
	CMPS	111	Introduction to Operating Systems	21	43	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Winter	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Spring	CMPS	223	Advanced Computer Security	12	83	no
	CMPS	223	Advanced Computer Security (LANL)	2	100	no
	CMPS	297A	Individual Study	2	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 136 of 411

	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	1	n/a	no

TEACHING 2009–10

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	5P	Introduction to Programming in Python	92	48	no
	CMPS	280S	Seminar on Computer Systems	16	75	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	297C	Individual Study	2	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Winter	CMPS	297A	Individual Study	2	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no
Spring	CMPS	198F	Independent Study or Research	1	n/a	no
	CMPS	229	Storage Systems	11	82	no
	CMPS	280S	Seminar on Computer Systems	14	36	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	3	n/a	no

TEACHING 2010–11

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	5C	Introduction to Programming in C/C++	119	45	no
	CMPS	280S	Seminar on Computer Systems	12	33	yes
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	3	n/a	no
Winter	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Spring	CMPS	111	Introduction to Operating Systems	28	64	no
	CMPS	280S	Seminar on Computer Systems	13	69	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	3	n/a	no

TEACHING 2011–12

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	198	Independent Study or Research	1	n/a	no
	CMPS	280S	Seminar on Computer Systems	20	40	yes
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Winter	CMPS	229	Storage Systems	13	38	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 137 of 411

	CMPS	297A	Individual Study	2	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Spring	CMPS	299C	Thesis Research	1	n/a	no
	CMPS	5P	Introduction to Programming in Python	136	66	no
	CMPS	280S	Seminar on Computer Systems	19	47	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	2	n/a	no
	CMPS	297B	Individual Study	2	n/a	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Summer	CMPS	297A	Individual Study	3	n/a	no
	CMPS	299	Thesis Research	1	n/a	no

TEACHING 2012–13

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	280S	Seminar on Computer Systems	17	76	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	2	n/a	no
	CMPS	299C	Thesis Research	6	n/a	no
Winter	CMPS	297A	Individual Study	2	n/a	no
	CMPS	297B	Individual Study	3	n/a	no
	CMPS	297C	Individual Study	2	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Spring	CMPS	280S	Seminar on Computer Systems	14	71	no
	CMPS	290X	Cryptgraph/Security	7	43	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
	CMPS	297B	Individual Study	3	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	6	n/a	no

TEACHING 2013–14

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	105	Systems Programming	18	22	no
	CMPS	280S	Seminar on Computer Systems	13	23	yes
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Summer	CMPS	297	Independent Study or Research	1	n/a	no
	CMPS	297A	Individual Study	1	n/a	no
Winter	CMPS	111	Introduction to Operating Systems	32	59	no
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	4	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 138 of 411

	EE	195	Senior Thesis Research	1	n/a	no
Spring	CMPS	280S	Seminar Computer Systems	14	50	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	2	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no

TEACHING 2014–15

				Enrolled	%Eval Retd	Shared?
Fall	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Winter	CMPS	111	Introduction to Operating Systems	42	69	no
	CMPS	280S	Seminar Computer Systems	11	45	yes
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	5	n/a	no
Spring	CMPS	223	Advanced Computer Security	7	86	no
	CMPS	195	Senior Thesis Research	1	n/a	no
	CMPS	280S	Seminar Computer Systems	11	45	yes
	CMPS	297B	Individual Study	1	n/a	no
	CMPS	299B	Thesis Research	1	n/a	no
	CMPS	299C	Thesis Research	6	n/a	no

TEACHING 2015–16

				Enrolled	%Eval Retd	Shared?
Summer	CMPS	299A	Thesis Research	1	n/a	no
Fall	CMPS	13H	Honors Introduction to Programming	40	68	no
	CMPS	13L	Honors Introduction to Programming Laboratory	17	n/a	no
	CMPS	13L	Honors Introduction to Programming Laboratory	19	n/a	no
	CMPS	280S	Seminar Computer Systems	9	44	yes
	CMPS	299C	Thesis Research	4	n/a	no
	DANM	297	Independent Study	1	n/a	no
Winter	CMPS	111	Introduction to Operating Systems	77	34	no
	CMPS	280S	Seminar Computer Systems	7	29	yes
	CMPS	299C	Thesis Research	3	n/a	no
Spring	CMPS	198F	Independent Study or Research	1	n/a	no
	CMPS	229	Storage Systems	9	89	no
	CMPS	280S	Seminar Computer Systems	14	64	yes
	CMPS	299C	Thesis Research	4	n/a	no

TEACHING 2016–17

				Enrolled	%Eval Retd	Shared?
Summer	CMPE	198	Independent Study or Research	1	n/a	no
	CMPS	193	Field Study	1	n/a	no
Fall	CMPS	296	Masters Project	1	n/a	no
	CMPS	299C	Thesis Research	2	n/a	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 139 of 411

Winter	CMPS	111	Introduction to Operating Systems	115	37	no
	CMPS	280S	Seminar Computer Systems	10	60	no
	CMPS	299C	Thesis Research	2	n/a	no
Spring	CMPE	297A	Individual Study	1	n/a	no
	CMPS	12B	Introduction to Data Structures	339	27	no
	CMPS	12M	Data Structures Laboratory	317	28	no
	CMPS	280S	Seminar Computer Systems	11	36	no

TEACHING 2017–18

				Enrolled	%Eval Retd	Shared?
Summer	CMPE	198	Individual Study or Research	1	n/a	no
Fall			Medical Leave			
Winter	CMPE	194F	Dangerous World	14	n/a	no
	CMPE	198F	Individual Study or Research	3	n/a	no
	CMPS	280S	Seminar Computer Systems	7	43	no
	CMPS	111	Introduction to Operating Systems	34	26	no
	CMPS	296	Masters Project	1	n/a	no
	CMPS	297A	Individual Study	3	n/a	no
	CMPS	297B	Individual Study	1	n/a	no
Spring	CMPE	236	Understanding Cryptography	10	100	no
	CMPE	296	Masters Project	2	n/a	no
	CMPE	297A	Individual Study	4	n/a	no
	CMPE	297B	Individual Study	2	n/a	no

TEACHING 2018–19

				Enrolled	%Eval Retd	Shared?
Summer	CMPE	198	Individual Study or Research	1	n/a	no
	CMPS	299	Thesis Research	1	n/a	no
Fall	CMPE	280S	Seminar Computer Systems	8	50	no
	CMPE	297A	Individual Study	2	n/a	no
	CMPE	297B	Individual Study	2	n/a	no
	CMPE	299A	Thesis Research	1	n/a	no
	CMPS	111	Introduction to Operating Systems	44	23	no
Winter	CMPE	195	Senior Thesis	4	n/a	no
	CMPS	198	Individual Study or Research	1	n/a	no
	CMPS	297A	Individual Study	2	n/a	no
	CMPS	297C	Individual Study	1	n/a	no
	CMPS	299A	Thesis Research	1	n/a	no
Spring	CMPS	280S	Seminar Computer Systems	11	36	no
	CMPE	296	Masters Project	1	n/a	no
	CMPE	297A	Individual Study	1	n/a	no
	CMPE	297B	Individual Study	1	n/a	no
	CMPE	299B	Thesis Research	1	n/a	no
	CMPE	299C	Thesis Research	1	n/a	no
	CMPS	111	Introduction to Operating Systems	154	23	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 140 of 411

TEACHING 2019–2020

				Enrolled	%Eval Retd	Shared?
Fall	CSE	13S	Computer Systems and C Programming	49	18	no
	CSE	280S	Seminar Computer Systems	11	27	no
Winter	CSE	13S	Computer Systems and C Programming	90	11	no
	CSE	280S	Seminar Computer Systems	11	27	no
Spring	CSE	234	Understanding Cryptography	23	52	no
	CSE	280S	Seminar Computer Systems	6	17	no

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 141 of 411

Mentoring and Advising

Advisor to Continuing Students

Degree Goal	Year Entered	Name
M.S.	2022	Eugene Chou
M.S.	2020	Sabrina Au
Ph.D.	2020	Kyle Fredrickson
Ph.D.	2018	Austen Barker
Ph.D.	2016	Oceane Bel
Ph.D.	2015	James Hughes
Ph.D.	2010	Michael M^{c}Throw

Postdoctoral Scholars

Name	Dates	Relationship	Degree Year	Institution
Yasu Ohara	2011 Fall–2013 Fall	Primary	2008	Keio University
Se Jin Kwon	2016 Spring–2017 Winter	Primary	2012	Ajou University

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 142 of 411

Doctor of Philosophy

Degree Year	Name	Relationship	Dissertation
2017	Joel Frank	Primary	Percival: A Reliable, Long-term, Distributed Storage System Free of Fixed-key Encryption
2017	Yan Li	Primary	ASCAR: Fully Autonomous Storage Contention Management System
2016	Stephanie Jones	Primary	Techniques for Reducing Long-term Data Movement on Shingled Magnetic Recording Drives
2016	Alexander Nelson	Primary	Software Signature Derivation From Sequential Digital Forensic Analysis
2016	Christina Strong	Primary	Data Allocation Approaches for Optimizing Storage Systems
2015	Alexandra Holloway	Primary	Design of a Data-Driven Micro-Display for Situation Awareness in Bursty Environments
2015	D. J. Capelis	Primary	Integrating Security into Lower Layers after Their Design
2013	Aleatha Parker-Wood	Primary	Improving File Management through Provenance and Rich Metadata
2010	Deepavali Bhagwat	Primary	Deduplication for Large Scale Backup and Archival Storage
2008	David Pease	Primary	Storage Tank: A Storage Area Network File System
2006	Lawrence You	Primary	Efficient Archival Data Storage
2005	Bo Hong	Co-Advisor	File and Storage Systems for MEMS-Based Storage
2002	Ahmed Amer	Primary	Predictive Data Grouping using Successor Prediction
2002	Tsozen Yeh	Primary	Improving File System Performance with File Access Predictions
2000	Theodore Haining	Primary	Non-volatile Cache Management for Improving Write Response Time with Rotating Magnetic Media
2000	Benjamin Reed	Primary	Using Network Attached Storage in a Secure Network Distributed File System
2000	Randal Burns	Primary	Data Management in a Distributed File System for Storage Area Networks
2000	Thomas Kroeger	Primary	Modeling File Access Patterns to Improve Caching Performance
1992	Richard Golding	Primary	Weak-consistency Group Communication and Membership

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 143 of 411

Doctor of Philosophy Dissertation Reading Committee

Degree Year	Name	Relationship	Advisor	Institution
—	Oceane Bel	Advisor	Prof. Darrell Long	UCSC
—	Devashish Purandare	Member	Prof. Ethan Miller	UCSC
—	James Hughes	Member	Prof. Darrell Long	UCSC
2019	Judith Samson	Member	Prof. Bradley Smith	UCSC
2019	Yuanjiang Ni	Member	Prof. Ethan Miller	UCSC
2019	Sinjoni Mukhopadhyay	Advisor	Prof. Darrell Long	UCSC
2019	James Byron	Member	Prof. Ethan Miller	UCSC
2019	Daniel Bittman	Member	Prof. Ethan Miller	UCSC
2015	Rekha Pitchumani	Member	Prof. Ethan Miller	UCSC
2015	D.J. Capelis	Member	Prof. Darrell Long	UCSC
2015	Alexandra Holloway	Advisor	Prof. Darrell Long	UCSC
2016	Raja Haddad	Président du jury	Witold Litwin	Université Paris–Dauphine
2015	Rekha Pitchumani	Member	Prof. Ethan Miller	UCSC
2014	Yangwook Kang	Member	Prof. Ethan Miller	UCSC
2014	Andrej Tolič	Member	Prof. Andrej Brodnik	University of Ljubljana
2013	Hanafi Yakouben	Membre du jury	Prof. Witold Litwin	Université Paris–Dauphine
2009	Karen Glocer	Member	Prof. Manfred Warmuth	UCSC
2009	Kevin Greenan	Member	Prof. Ethan Miller	UCSC
2009	Mark Storer	Member	Prof. Ethan Miller	UCSC
2007	Timothy Bisson	Member	Prof. Scott Brandt	UCSC
2007	John Colby	Member	Prof. Geoffrey Pullum	UCSC
2007	Dean Hildebrand	Member	Prof. Peter Honeyman	University of Michigan
2006	Zachary Peterson	Member	Prof. Randal Burns	Johns Hopkins University
2006	Feng Wang	Member	Prof. Scott Brandt	UCSC
2006	Caixue Lin	Member	Prof. Scott Brandt	UCSC
2005	Reiner Kraft	Member	Prof. Raymie Stata	UCSC
2005	Qin Xin	Member	Prof. Ethan Miller	UCSC
2005	Scott Banachowski	Member	Prof. Scott Brandt	UCSC
2004	Ismail Ari	Member	Prof. Ethan Miller	UCSC
2004	Dmitrii Zagorodnov	Member	Prof. Keith Marzullo	UCSD
2003	R. Kameshwari	Member	Prof. K. Gopinath	IIS Bangalore
2003	Bradley Smith	Member	Prof. J.J. Garcia-Luna	UCSC
2002	Brian Chess	Member	Prof. F. Joel Ferguson	UCSC
2002	Jyoti Raju	Member	Prof. J.J. Garcia-Luna	UCSC
1999	Rodrigo Garcés	Member	Prof. J.J. Garcia-Luna	UCSC
1999	Clay Shields	Member	Prof. J.J. Garcia-Luna	UCSC
1998	Bruce Montague	Member	Prof. Charles M^{c}Dowell	UCSC
1996	Shree N. Murthy	Member	Prof. J.J. Garcia-Luna	UCSC
1996	Eric Rosen	Member	Prof. Patrick Mantey	UCSC
1995	Yuxia Zhang	Member	Prof. J.J. Garcia-Luna	UCSC
1993	Daniel Edelson	Member	Prof. Ira Pohl	UCSC
1993	Max Copperman	Member	Prof. Charles M^{c}Dowell	UCSC

Doctor of Philosophy Qualifying Committee

Year	Name	Relationship	Advisor	Institution
2020	Shreyas Kumar	Member	Prof. Darrell Long	UCSC

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 144 of 411

2020	Devashish Purandare	Chair	Prof. Ethan Miller	UCSC
2020	James Hughes	Chair	Prof. Darrell Long	UCSC
2018	Judith Samson	Member	Prof. Bradley Smith	UCSC
2018	Trevor Pesout	Outside Member	Prof. Darrell Long	UCSC
2018	Yuanjing Ni	Chair	Prof. Ethan Miller	UCSC
2018	Sinjoni Mukhopadhyay	Member	Prof. Darrell Long	UCSC
2018	James Byron	Chair	Prof. Ethan Miller	UCSC
2018	Daniel Bittman	Chair	Prof. Ethan Miller	UCSC
2018	Oceane Bel	Member	Prof. Darrell Long	UCSC
2014	Preeti Gupta	Chair	Prof. Ethan Miller	UCSC
2012	Yangwook Kang	Chair	Prof. Ethan Miller	UCSC
2012	Rekha Pitchumani	Chair	Prof. Ethan Miller	UCSC
2012	Ian Adams	Chair	Prof. Ethan Miller	UCSC
2011	Avani Wildani	Chair	Prof. Ethan Miller	UCSC
2008	Neerja Bhatnagar	Chair	Prof. Ethan Miller	UCSC
2008	Kevin Greenan	Chair	Prof. Ethan Miller	UCSC
2008	Andrew Leung	Chair	Prof. Ethan Miller	UCSC
2007	Purvi Shah	Member	Prof. Jehan-François Pâris	University of Houston
2007	Mark Storer	Chair	Prof. Ethan Miller	UCSC
2006	David Essary	Member	Prof. Ahmed Amer	University of Pittsburgh
2005	Karen Glocer	Member	Prof. Manfred Warmuth	UCSC
2004	Reiner Kraft	Chair	Prof. Raymie Stata	UCSC
2003	Scott Banachowski	Chair	Prof. Scott Brandt	UCSC
2003	Qin Xin	Chair	Prof. Ethan Miller	UCSC
2003	Lawrence You	Member	Prof. Darrell Long	UCSC
2003	Feng Wang	Chair	Prof. Scott Brandt	UCSC
2003	Dmitrii Zagorodnov	Member	Prof. Keith Marzullo	UCSD
2003	Bo Hong	Member	Prof. Darrell Long	UCSC
2002	Ismail Ari	Chair	Prof. Ethan Miller	UCSC
2002	Bradley Smith	Chair	Prof. J.J. Garcia-Luna	UCSC
2001	Tsozen Yeh	Member	Prof. Darrell Long	UCSC
2001	Aman Shaikh	Member	Prof. Anujan Varma	UCSC
2000	Brian Chess	Chair	Prof. F. Joel Ferguson	UCSC
2000	Ahmed Amer	Member	Prof. Darrell Long	UCSC
1999	Srinivas Vutukury	Member	Prof. J.J. Garcia-Luna	UCSC
1999	Jyoti Raju	Member	Prof. J.J. Garcia-Luna	UCSC
1998	Marcelo Spohn	Member	Prof. J.J. Garcia-Luna	UCSC
1998	Gil Fuchs	Member	Prof. Robert Levinson	UCSC
1998	Benjamin Reed	Member	Prof. Darrell Long	UCSC
1997	Thomas Kroeger	Member	Prof. Darrell Long	UCSC
1997	Thomas Clay Shields	Member	Prof. J.J. Garcia-Luna	UCSC
1997	Steven Carter	Member	Prof. Darrell Long	UCSC
1997	Brian Levine	Member	Prof. J.J. Garcia-Luna	UCSC
1997	Brian Chess	Member	Prof. F. Joel Ferguson	UCSC
1996	Lampros Kalampoukas	Member	Prof. J.J. Garcia-Luna	UCSC
1996	Hans-Peter Dommel	Member	Prof. J.J. Garcia-Luna	UCSC
1996	Fumiaki Kamiya	Member	Prof. Allen Van Gelder	UCSC
1996	Dirk Coldeway	Member	Prof. Charles M^{c}Dowell	UCSC
1996	Bruce Montague	Member	Prof. Charles M^{c}Dowell	UCSC
1995	Theodore Haining	Member	Prof. Darrell Long	UCSC
1995	Shree N. Murthy	Member	Prof. J.J. Garcia-Luna	UCSC

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 145 of 411

1995	Jochen Behrens	Member	Prof. J.J. Garcia-Luna	UCSC
1995	Dimitrios Stiliadis	Member	Prof. Anujan Varma	UCSC
1995	Chane Fullmer	Member	Prof. J.J. Garcia-Luna	UCSC
1994	Gil Fuchs	Member	Prof. Robert Levinson	UCSC
—	Tsozen Yeh	Member	Prof. Darrell Long	UCSC
—	Thomas Shields	Member	Prof. J.J. Garcia-Luna	UCSC
—	Steven Carter	Member	Prof. Darrell Long	UCSC
—	Shree Murthy	Member	Prof. J.J. Garcia-Luna	UCSC
—	Sage Weil	Member	Prof. Scott Brandt	UCSC
—	Richard Golding	Member	Prof. Darrell Long	UCSC
—	Randal Burns	Member	Prof. Darrell Long	UCSC
—	Max Copperman	Member	Prof. Charles M^{c}Dowell	UCSC
—	Marcelo Spohn	Member	Prof. J.J. Garcia-Luna	UCSC
—	Lawrence You	Member	Prof. Darrell Long	UCSC
—	Kjell Post	Member	Prof. Allen Van Gelder	UCSC
—	Ignacio Solis	Member	Prof. J.J. Garcia-Luna	UCSC
—	Ignacio Corderi	Member	Prof. Darrell Long	UCSC
—	Gil Fuchs	Member	Prof. Robert Levinson	UCSC
—	Fumiaki Okushi	Member	Prof. Allen Van Gelder	UCSC
—	Eric Rosen	Member	Prof. Patrick Mantey	UCSC
—	Dirk Coldewey	Member	Prof. Charles M^{c}Dowell	UCSC
—	Dimitrios Stiliadis	Member	Prof. J.J. Garcia-Luna	UCSC
—	David Pease	Member	Prof. Darrell Long	UCSC
—	Chane Fullmer	Member	Prof. J.J. Garcia-Luna	UCSC
—	Brian Levine	Member	Prof. J.J. Garcia-Luna	UCSC
—	Bo Hong	Member	Prof. Scott Brandt	UCSC
—	Benjamin Reed	Member	Prof. Darrell Long	UCSC
—	Aman Shaikh	Member	Prof. J.J. Garcia-Luna	UCSC
—	Ahmed Amer	Member	Prof. Darrell Long	UCSC

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 146 of 411

Master of Science

Degree Year	Name	Relationship	Thesis
2015	Ignacio Corderi	Primary	RESAR: A System for a Two-Failure Tolerant, Self-Adjusting Million Disk Storage Cluster
2014	Erik Steggall	Primary	Sleuthkit: XTAF extension
2014	Michael M^{c}Throw	Primary	Extended Metadata and File System Design: Designing File Systems that Take Advantage of Rich Metadata
2013	Sonali Somyalipi	Primary	Impact of Differential Compression and Deduplication on Genomic Data Storage
2012	Akhilesh Bhargav Malavalli	Primary	Improved Measurement of Backup Reconstruction Performance for a Deduplicated System
2010	Stephanie Jones	Primary	On-line Deduplication in a Log-Structured File System for Primary Storage
2008	Rosie Wacha	Primary	Data Reliability Techniques for Specialized Storage Environments
2007	Kristal Pollack	Primary	Quota Enforcement for High-Performance Distributed Storage Systems
2005	Jeffrey Rybczynski	Primary	Active Workload Reshaping to Maximize Disk Energy Conservation
2005	Jörg Meyer	Primary	Large Scale Multi-Type Inverted List Indexing
2004	Svetlana Kagan	Primary	Using Containers for Efficient Storage of Small Files in the Deep Store Archival System
2004	Karl Brandt	Primary	Using Multiple Experts to Perform File Prediction
2003	Ian Brown	Primary	Upgrading NFS File Servers in a Live Environment
2003	Karen Glocer	Primary	The Utility of Fine-Grained Workload Characterizations
2002	Ying Lin	Primary	Power Conservation Strategies for MEMS-Based Storage Devices
2002	Zachary Peterson	Primary	Data Placement for Copy-on-Write Using Virtual Contiguity
2000	Pu Yang	Primary	Modeling Probe Based Data Storage Devices
2000	A. David M^{c}Nab	Primary	Extensible UNIX Access Control Lists
1999	John Colby	Primary	Error Correction for Incremental LR Parsers
1998	Tracey Sconyers	Primary	An Examination of Power Consumption in Wireless Modems
1997	Randal Burns	Primary	Differential Compression: A Generalized Solution for Binary Files
1996	Thomas Kroeger	Primary	Predicting File System Actions From Reference Patterns
1994	David Schreiber	Primary	A Code Generator for the PowerPC 601
1994	Chane Fullmer	Primary	Implementation of Adaptive Flow Control Mechanism to the Swift Redundant Distributed File Architecture
1993	Michelle Abram	Primary	Transparent Remote Procedure Calls
1993	Kulumani B. Sriram	Primary	A Study of the Reliability of Hosts on the Internet
1993	Cheng Tang	Primary	A Transport Mechanism for Supporting Multimedia Traffic in ATM Networks
1992	William Osser	Primary	Automatic Process Selection for Load Balancing
1992	Carol Osterbrock	Primary	A Transport Level Protocol for Guaranteed Performance in a Local Area Network
1991	Richard Golding	Primary	Accessing Replicated Data in a Large Scale Distributed System

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 147 of 411

Master of Science Thesis Reading Committee

Year	Name	Relationship	Advisor
2018	Isaak Cherdak	Chair	Prof. Ethan Miller
2018	Matthew Bryson	Member	Prof. Ethan Miller
2018	Oceane Bel	Chair	Prof. Darrell Long
2018	Austen Barker	Chair	Prof. Darrell Long
2017	Preeti Gupta	Chair	Prof. Darrell Long
2015	Xiaoyuan Lu	Member	Prof. Darrell Long
2015	Ignacio Corderi	Chair	Prof. Darrell Long
2011	Nathan Edel	Member	Prof. Ethan Miller
2010	Kiren Jin	Member	Prof. Ethan Miller
2004	Narayan Brooks	Member	Prof. Suresh Lodha
2003	David Pease	Member	Prof. Richard Hughey
2002	Reiner Kraft	Member	Prof. Raymie Stata
2002	Feng Wang	Member	Prof. Scott Brandt
2002	Alicia Szczurowska	Member	Prof. Scott Brandt
2000	Paul Caamano	Member	Prof. Charles M^{c}Dowell
1999	Manosiz Bhattacharrya	Member	Prof. Charles M^{c}Dowell
1998	Christina Parsa	Member	Prof. J.J. Garcia-Luna
1998	Jyoti Raju	Member	Prof. J.J. Garcia-Luna
1997	Bradley Smith	Member	Prof. J.J. Garcia-Luna
1997	Michael Allen	Member	Prof. Charles M^{c}Dowell
1997	Bruce Sherrod	Member	Prof. David Helmbold
1996	Brian Levine	Member	Prof. J.J. Garcia-Luna
1996	Thomas Clay Shields	Member	Prof. J.J. Garcia-Luna
1996	Catherine Wilson	Member	Prof. Suresh Lodha
1995	Brian Chess	Member	Prof. F. Joel Ferguson
1994	Shree N. Murthy	Member	Prof. J.J. Garcia-Luna
1993	Theodore Haining	Member	Prof. Charles M^{c}Dowell
1993	Rock Pfotenhauer	Member	Prof. Ira Pohl
1993	Richard Snyder	Member	Prof. Robert Levinson
1991	Matthew Breaden	Member	Prof. Tracy Larrabee
1990	Max Copperman	Member	Prof. Charles M^{c}Dowell
1990	Daniel Edelson	Member	Prof. Ira Pohl
1990	Frank Jas	Member	Prof. Karl Schimpf
1989	James Kerr	Member	Prof. Allen Van Gelder

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 148 of 411

Bachelor of Science

Degree Year	Name	Relationship	Thesis
2019	Yash Gupta	Primary	Artifice: A Deniable Steganographic File System
2016	Daniel Bittman	Primary	Designing a Wait-free MPSC Queue
2011	Sulimon Sattari	Primary	A Cross-Disciplinary Data Sharing and Querying Platform for Earth Science Data
2001	Christopher Benson	Primary	Peer-to-Peer File System using Weak Consistency Replication
1994	Sean Gilligan	Primary	MacMet: A Macintosh Program to Display Real-Time and Historical Data from Meteorological Stations
1992	Nitin Ganatra	Primary	Census: Collecting Host Information on a Wide Area Network
1991	Aaron Emigh	Primary	The Swift Architecture: Anatomy of a Prototype
1989	Leland Wallace	Primary	Font Extensions to the **X**-Fig Graphics Program

US 20130282787A1

(19) **United States**

(12) **Patent Application Publication**

Lippert

(10) **Pub. No.: US 2013/0282787 A1**

(43) **Pub. Date: Oct. 24, 2013**

(54) **COMPUTER CLUSTER ARRANGEMENT FOR PROCESSING A COMPUTATION TASK AND METHOD FOR OPERATION THEREOF**

(71) Applicant: **ParTec Cluster Competence Center GmbH**, Munchen (DE)

(72) Inventor: **Thomas Lippert**, Aschaffenburg (DE)

(21) Appl. No.: **13/861,429**

(22) Filed: **Apr. 12, 2013**

Related U.S. Application Data

(63) Continuation of application No. PCT/EP2011/067888, filed on Oct. 13, 2011.

(30) **Foreign Application Priority Data**

Oct. 13, 2010 (EP) 10187436.0

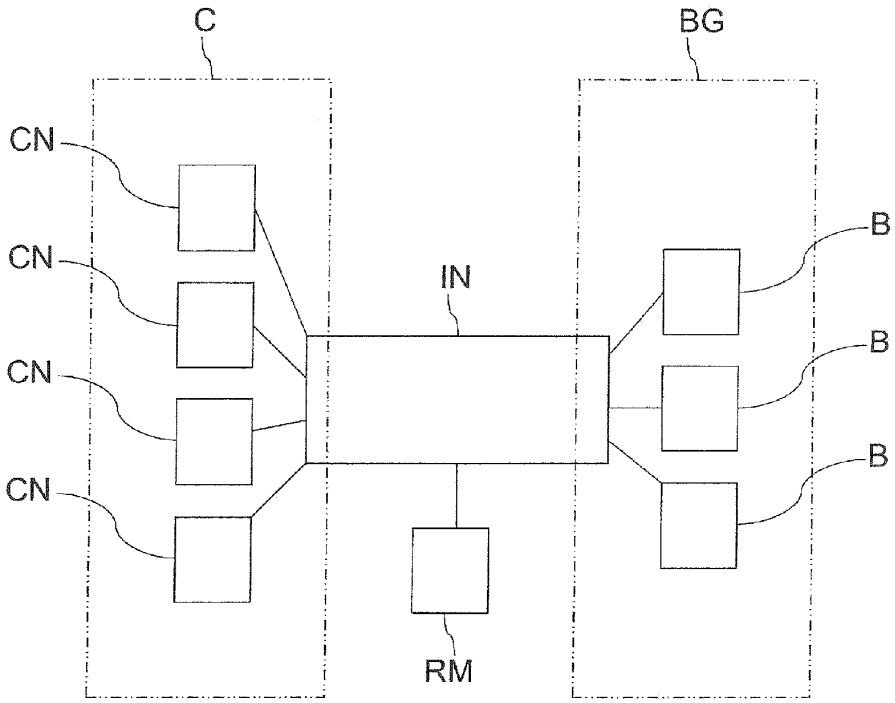
Publication Classification

(51) **Int. Cl.**
H04L 29/08 (2006.01)

(52) **U.S. Cl.**
CPC ***H04L 29/08081*** (2013.01)
USPC **709/201**

(57) **ABSTRACT**

The present invention is directed to a computer cluster arrangement and a method for operation of the introduced computer cluster arrangement. The computer cluster arrangement comprises computation nodes CN, which dynamically outsource specific computation tasks to boosters B. Therefore, an assignment technique of boosters B to computation nodes CN is introduced. The assignment takes dynamically place at runtime. The present invention finds application in high performance cluster technology.



Darell Long, Ph.D.
09/17/2025
Exhibit C
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1004

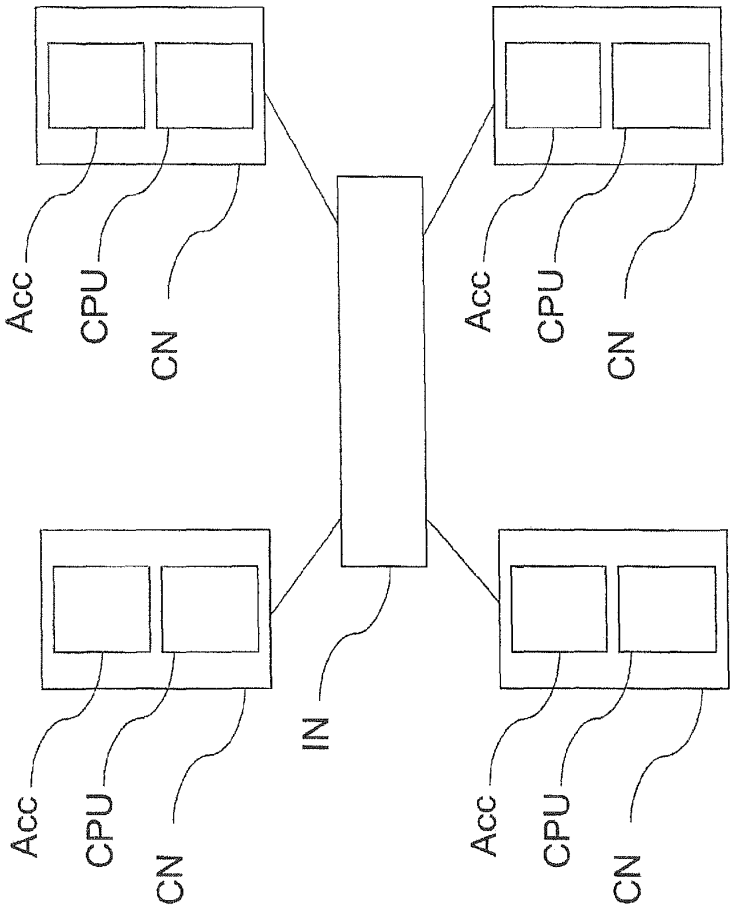


Fig.: 1

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 151 of 411

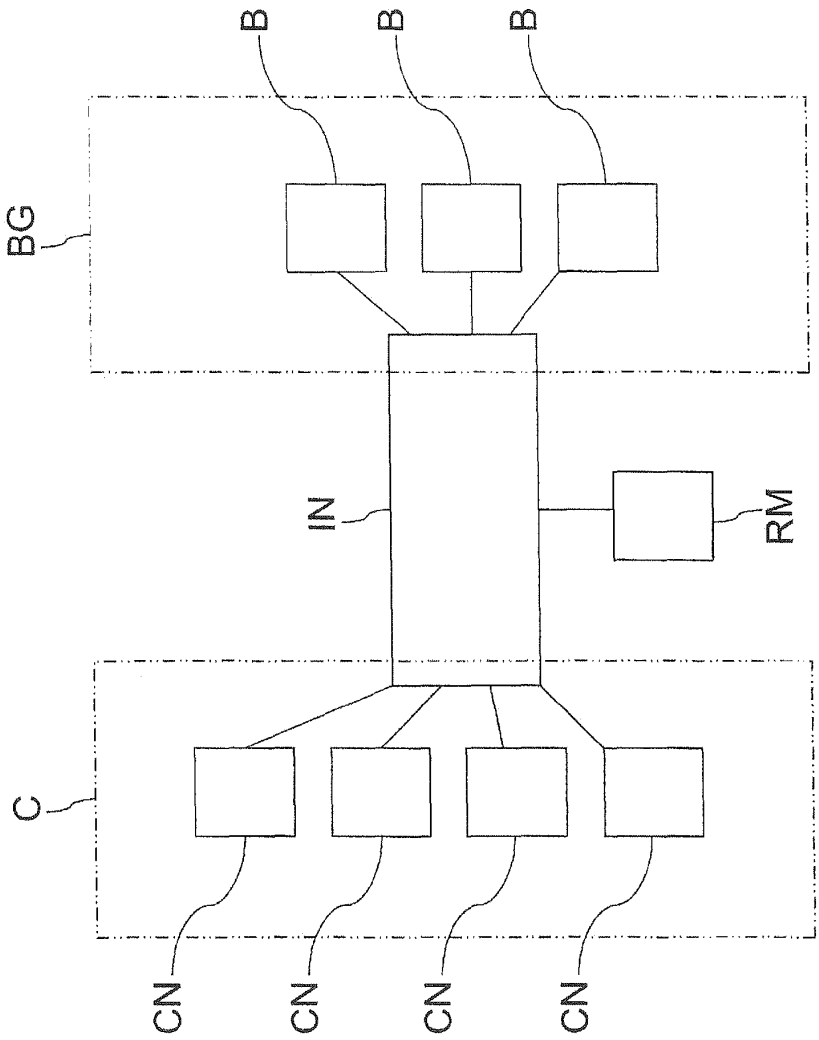


Fig.: 2

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 152 of 411

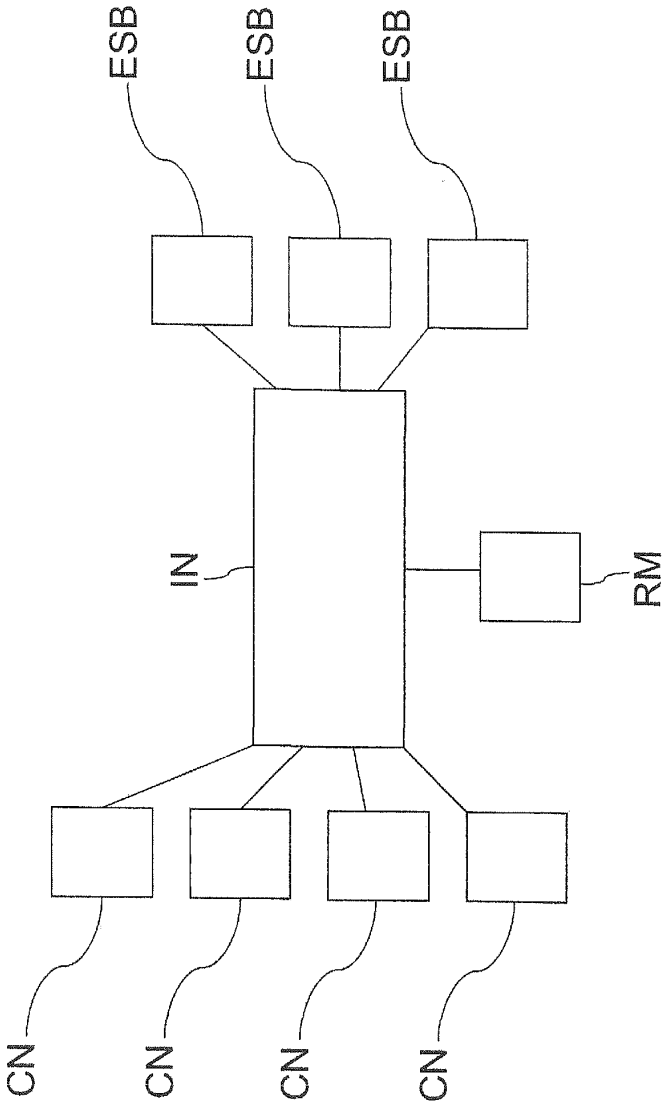


Fig.: 3

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 153 of 411

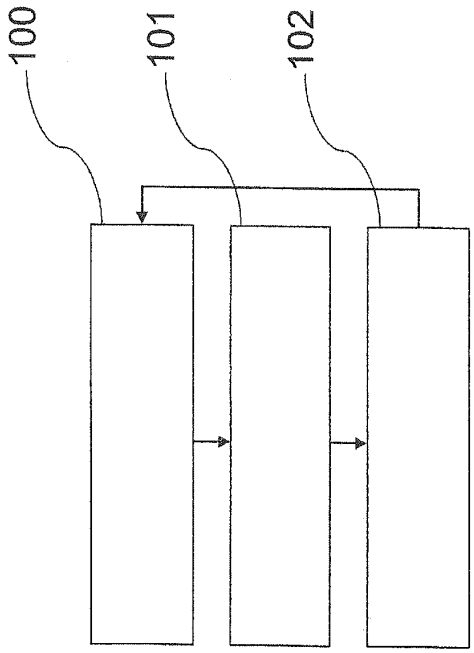


Fig.: 4

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 154 of 411

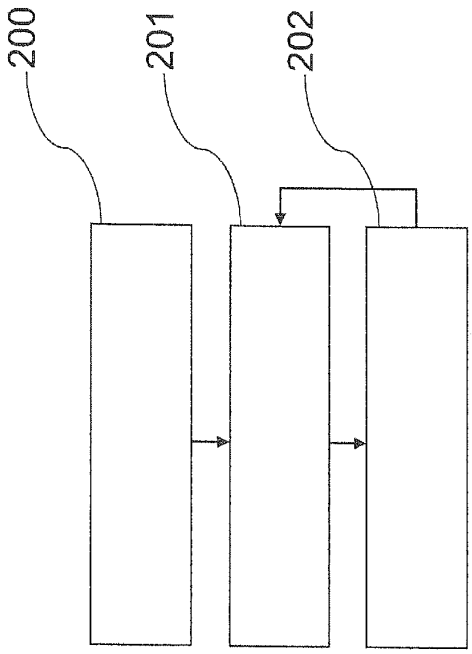


Fig.: 5

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 155 of 411

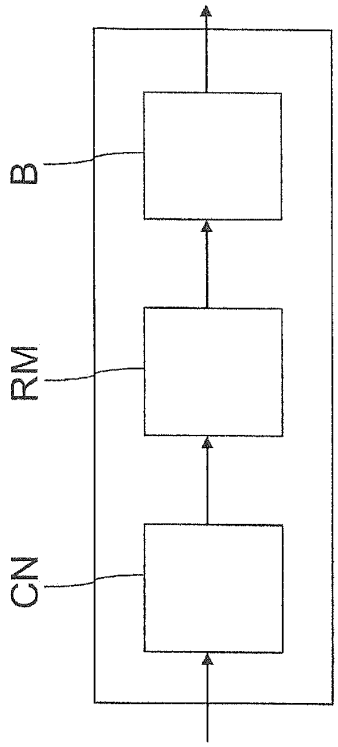


Fig.: 6

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 156 of 411

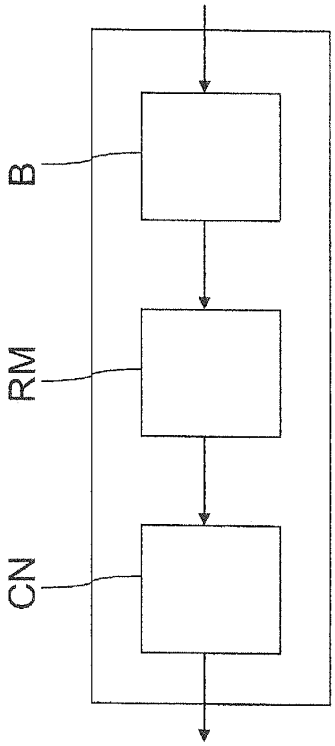


Fig.: 7

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 157 of 411

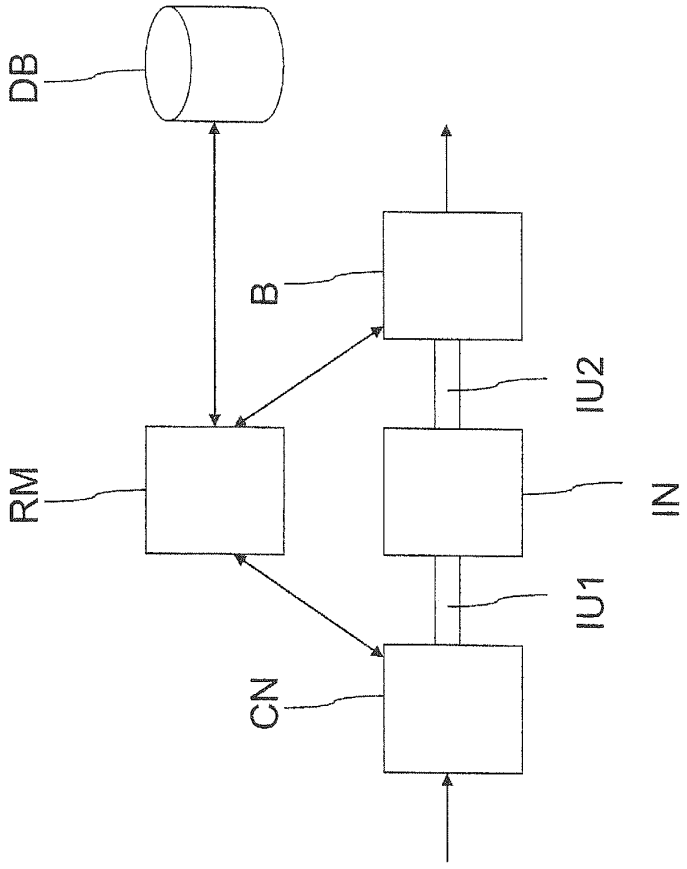


Fig.: 8

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 158 of 411

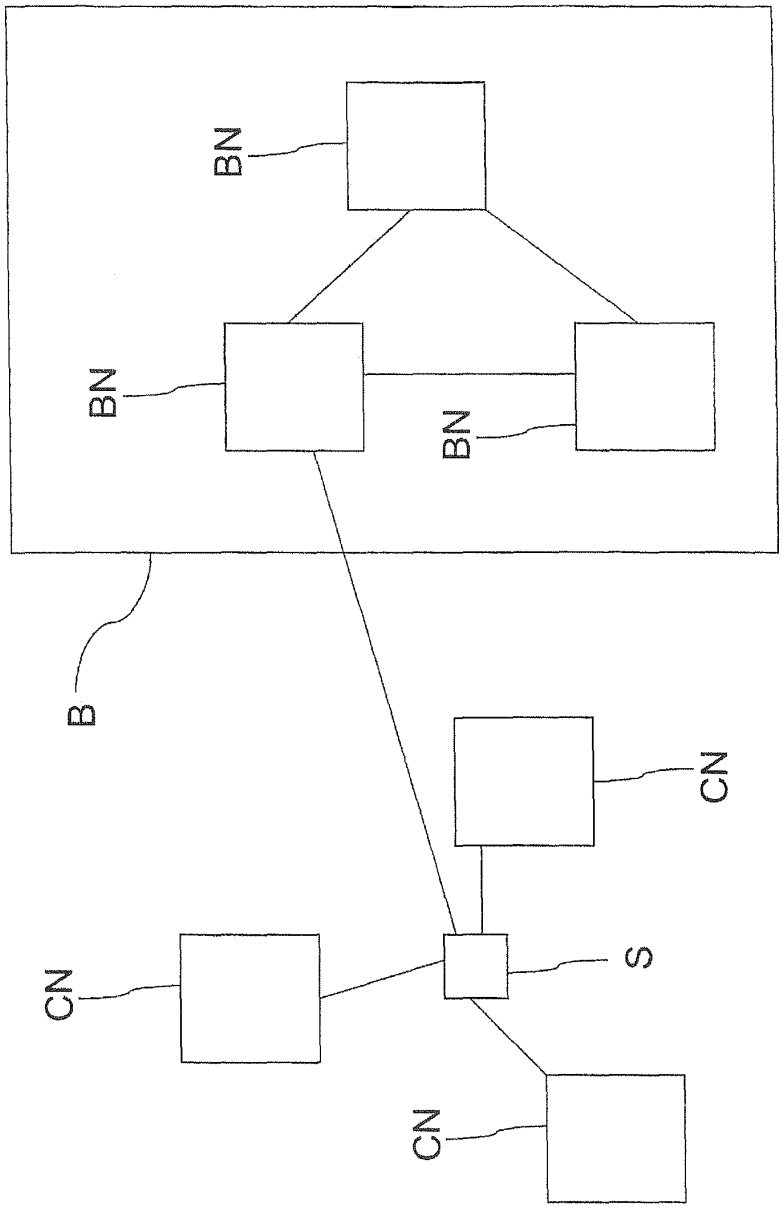


Fig.: 9

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 159 of 411

COMPUTER CLUSTER ARRANGEMENT FOR PROCESSING A COMPUTATION TASK AND METHOD FOR OPERATION THEREOF

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of PCT Application No. PCT/EP2011/067888 filed 13, Oct. 2011, which was published in the English language on 19, Apr. 2012 as PCT Publication No. WO 2012/049247. The PCT Application claims priority to EP Application No. 10187436.0 filed 13, Oct. 2010. These documents are hereby incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention is directed toward a computer cluster arrangement. In particular, it relates to a computer cluster arrangement with improved resource management as regards the application of computing nodes for processing scalable computation tasks as well as complex computation tasks. It is especially directed toward a computer cluster arrangement for processing a computation task and a method for operating the computer cluster arrangement. The computer cluster arrangement in accordance with the present invention makes use of acceleration functionality, which assist the computing nodes to accomplish a given computation task. The present invention is furthermore directed toward a computer program product being configured for accomplishing the method as well as a computer readable medium for storing the computer program product.

BACKGROUND OF THE INVENTION

[0003] Known in the art are computer cluster arrangements comprising computing nodes including at least one processor as well as accelerators being tightly coupled to the computing nodes for outsourcing computations of high resource requirements. A tight coupling of accelerators to computing nodes results in a static assignment and leads to over- or under-subscription of accelerators. This may lead to a lack of resources or may lead to an excessive supply of resources. Such a static assignment of accelerators to computing nodes does furthermore not provide fault tolerance in case of accelerator failures.

[0004] The publication "rCUDA: reducing the number of GPU-based accelerators in high performance clusters" by Jose Duato, Rafael Mayo et al., International Conference on High Performance Computing and Simulation (HPCS), Issue Date: Jun. 28, 2010-Jul. 2, 2010, on pages 224-231, describes a frame work that enables remote GPU acceleration in high performance clusters, thus allowing a reduction in the number of accelerators installed on the cluster. This may lead to energy, acquisition, maintenance and space savings.

[0005] The publication "A package for open CL based heterogeneous computing on clusters with many GPU devices" by Amnon Barak, et al. of the Department of Computer Science from Hebrew University of Jerusalem describes a package for running OpenMP, C++ an unmodified OpenCL applications on clusters with many GPU devices. Furthermore, an implementation of the OpenCL specifications and extensions of the OpenMP API that allow applications on one hosting-node to transparently utilize cluster-wide devices is provided. FIG. **1** shows a computer cluster arrangement according to the state of the art. The computer cluster arrangement comprises several computations nodes CN, which are interconnected and jointly compute a computation task. Each computation node CN is tightly coupled with an accelerator Acc. As can be seen in FIG. **1** a computation node CN comprises an accelerator unit ACC which is virtually integrated on the computation node CN along with a microprocessor, for instance a central processing unit CPU. As introduced above, the fixed coupling of accelerators Acc to computation nodes CN leads to an over- or under subscription of accelerators Acc depending on the computation task. Furthermore, no fault tolerance is provided in case of failure of one of the accelerators Acc. In the known computer cluster arrangement according to FIG. **1** computing nodes CN communicate with each other over an infrastructure, wherein accelerators Acc do not exchange information directly, but require a computation node CN interfacing the infrastructure IN for data exchange.

SUMMARY OF THE INVENTION

[0006] Hence, it is an object of the present invention to provide a computer cluster arrangement, which allows communication flexibility as regards data exchange between accelerator and computation nodes as well as direct access of computation nodes to any and each of the accelerators.

[0007] Furthermore, it is an object of the present invention to provide a dynamic coupling of accelerators to computation nodes at runtime.

[0008] These objects are solved by a computer cluster arrangement with features according to PCT patent claim 1 (presented below).

[0009] Accordingly, a computer cluster arrangement for processing a computation task is provided, the computer cluster arrangement comprising:

- **[0010]** a plurality of computation nodes, each of which interfacing a communication infrastructure, at least two of which being arranged to jointly compute at least a first part of the computation task;
- **[0011]** at least one booster being arranged to compute at least a second part of the computation task, each booster interfacing the communication infrastructure; and
- **[0012]** a resource manager being arranged to assign at least one booster to at least one of the plurality of computation nodes for computation of the second part of the computation task, the assignment being accomplished as a function of a predetermined assignment metric.

[0013] In this computer cluster arrangement acceleration functionality is being provided by independent boosters. The described computer cluster arrangement allows a loose coupling of those boosters to computation nodes, which may also be referred to as compute nodes. Hence, a sharing of accelerators, here in form of boosters, by computation nodes is feasible. For an assignment of a booster to a computation node a resource manager, in form of a resource manager module or resource manager node, may be provided. The resource manager may establish a static assignment at start of a processing of a computation task. Alternatively or additionally it may establish a dynamic assignment at runtime, which means during processing of the computation task.

[0014] The resource manager is arranged to provide assignment information to the computation nodes for outsourcing parts of the computation tasks from at least one computation node to at least one booster. The resource manager may be implemented as a specific hardware unit, a virtual unit or be compound of any of them. Especially the resource manager may be formed by any-one of: a microprocessor, a hardware

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 160 of 411

component, a virtualized hardware component or a daemon. Furthermore parts of the resource manager may be distributed over the system and communicate via a communication infrastructure.

[0015] Communication between boosters is accomplished through a network protocol. Hence, booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task. Fault tolerance in case of booster failure is provided as well as scalability is fostered. Scalability is made possible by support of incremental system development, as boosters are provided independently of computation nodes. Hence, the number of computation nodes and the number of provided boosters may differ. Thus, a maximum flexibility in providing hardware resources is established. Furthermore, all computation nodes do share the same growth capacity.

[0016] A computation task may be defined by means of an algorithm, a source code, a binary code and may be furthermore be compound of any of them. A computation task may for instance be a simulation, which is to be computed by the computer cluster arrangement. Furthermore, the computation task may comprise several sub problems, also referred to as sub tasks, which in their entirety describe the overall computation task. It is possible to divide the computation task into several parts, for instance at least a first part of the computation task and at least a second part of the computation task. It is also possible for the computer cluster arrangement to solve the parts of the computation task in parallel or in succession. Each computation node interfaces a communication infrastructure, also referred to as interconnect. Analogously, each booster interfaces the communication infrastructure. Hence, the computation nodes as well as the boosters interact by means of the communication infrastructure. Therefore, each computation node communicates with each booster over the communication infrastructure, without the necessity to involve a further communication node while exchanging data from a computation node to a booster. Thus, a dynamic assignment of computation nodes to boosters is established, wherein computation nodes process at least a part of the computation task and are not required for passing through of information from one computation node to one booster. Therefore, it is possible to directly couple boosters to the communication infrastructure without the necessity of an intermediate computation node as it is typically implemented in the state of the art.

[0017] For accomplishing the assignment between boosters and computation nodes a specific set of rules is required. Therefore, an assignment metric is provided, which serves as a basis for the decision which booster is coupled with which computation node. The assignment metric may be managed by a resource manager. Managing the assignment metric refers to establishing and updating rules naming at least one booster, which is assigned to at least one further named computation node. Hence, it is possible to update the assignment metric at runtime. Such assignment rules may be created as a function of a load balancing, which detects workload of the computer cluster arrangement, especially of the boosters. Furthermore, it is possible to detect computing capacities of boosters and furthermore detect computation task requirements and assign a selected booster, which provides the required capacities to the computation node. For determining an initial assignment of boosters to computation nodes, the assignment metric is predetermined but may be altered at runtime. Hence, static assignment is provided at start of the processing of computation task and dynamic assignment is provided at runtime.

[0018] In an embodiment of the present invention the determined assignment metric is formed according to at least one of group of metric specification techniques, the group comprising: a temporal logic, an assignment matrix, an assignment table, a probability function and a cost function. Hence, temporal dependencies may be considered for assigning the boosters. It may be the case, that a temporal order is defined on the boosters, which makes sure that a specific booster is always assigned to a computation node in case a further booster failed to solve at least a part of the computation task. Hence, a hierarchy between boosters can be considered for their assignment. An assignment metric may name an identification of a computation node and may furthermore define identifications of compatible boosters which can be assigned. A probability function may for instance describe that in case a specific booster failed to compute a certain computation task a further booster may solve the same computation task at a specific probability. Furthermore cost functions may be applied for evaluation of required resource capacities and furthermore for evaluation of provided computation capacities of boosters.

[0019] Hence, computation tasks of certain requirements can be forwarded to appropriate boosters.

[0020] A computation history, also referred to as computation log record, may also be applied for dynamic assignment. Hence, computation tasks can be empirically evaluated by computation on at least one first booster and recording response times and furthermore by processing the same computation task on at least one further booster and recording response times. Hence, capacities of boosters can be recorded, empirically evaluated and therefore be assigned to computation nodes as a function of required capacities and their provided capacities. Specific computation tasks may comprise priority information, which indicates how urgently this specific computation task has to be computed. It may also be the case that specific computation nodes provide a priority, which indicates how urgent a processing of a computation task, or at least a part of a computation task, is compared to other parts of computation tasks being originated from other computation nodes. Hence, it is possible to provide priority information as regards single parts of the computation task as well as priority information referring to computation nodes. Once a booster is assigned to a computation node, the booster processes specific parts of a computation task. This may be accomplished by a remote procedure call, a parameter handover or data transmission. The complexity of the part of the computation task may be evaluated as a function of a parameter handover. In case a parameter contains a matrix, the complexity of the parameter handover can be evaluated by the number of dimensions of the matrix.

[0021] For interfacing the communication infrastructure an interfacing unit may be provided, which is arranged between one computation node and the communication infrastructure. A further interfacing unit being different from the first interfacing unit, may be arrange between the booster and the communication infrastructure. The interfacing unit can be different form the computation node and is also different from the booster. The interfacing unit merely provides network functionality, without being arranged to process parts of the computation task. The interfacing unit merely provides functionality as regards the administration and communication

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 161 of 411

issues of the computation tasks. It may for example provide functionality as regards routing and transmission of data referring to the computation task. Furthermore, acceleration can also be performed reversely by outsourcing at least a part of the computation task from at least one booster to at least one computation node. Hence, control and information flow is reversed as regards the above introduced aspects of the invention.

[0022] According to an aspect of the present invention, the predetermined assignment may be formed according to at least one group of matrix specification techniques, the group comprising: a temporal logic, an assignment matrix, an assignment table, a probability function and a cost function. This may provide the advantage that the predetermined assignment metric may be formed under usage of a formal or semiformal model or data type.

[0023] According to a further aspect of the present invention, the predetermined assignment metric is specified as a function of at least one of a group of assignment parameters, the group comprising: resource information, cost information, complexity information, scalability information, a computation log record, compiler information, priority information and a time stamp. This may provide the advantage that the assignment may be performed dynamically at runtime under consideration of different runtime parameters and in response to specific computation task characteristics.

[0024] According to a further aspect of the present invention, the assignment of at least one booster to one of the plurality of computation nodes triggers at least one of a group of signals, the group comprising: a remote procedure call, a parameter handover and a data transmission. This may provide the advantage that at least a part of the computation tasks can be forwarded from one computation node to at least one booster.

[0025] According to a further aspect of the present invention, each computation node and each booster interfaces the communication infrastructure respectively via an interfacing unit. This may provide the advantage that data can be communicated via the communication infrastructure without the necessity of an intermediate computation node. Hence, it is not required to couple a booster with a computation node directly but a dynamic assignment is reached.

[0026] According to a further aspect of the present invention, the interfacing unit comprises at least one group of components, the group comprising: a virtual interface, a stub, a socket, a network controller and a network device. This may provide the advantage that the computation nodes as well as the boosters can also be virtually connected to the communication and infrastructure. Furthermore existing communication infrastructures can be easily accessed. According to a further aspect of the present invention, communication and infrastructure comprises at least one of a group of components, the group comprising: a bus, a communication link, a switching unit, a router and a high speed network. This may provide the advantage that existing communication infrastructures can be used and new communication infrastructures can be created by commonly available network devices.

[0027] According to a further aspect of the present invention, each computation node comprises at least one of a group of components, the group comprising: a multi core processor, a cluster, a computer, a workstation and a multipurpose processor. This may provide the advantage that the computation nodes are highly scalable.

[0028] According to a further aspect of the present invention, the at least one booster comprises at least one group of components, the group comprising: a many-core-processor, a scalar processor, a co-processor, a graphical processing unit, a cluster of many-core-processors and a monolithic processor. This may provide the advantage that the boosters are implemented to process specific problems at high speed.

[0029] Computation nodes typically apply processors comprising an extensive control unit as several computation tasks have to be processed simultaneously. Processors being applied in boosters typically comprise an extensive arithmetic logic unit and a simple control structure when being compared to computation nodes processors. For instance SIMD, also refer to as single instruction multiple data computers, may find application in boosters. Hence, processors being applied in computation nodes differ in their processor design compared to processors being applied in boosters.

[0030] According to a further aspect of the present invention, the resource manager is arranged to update said predetermined assignment metric during computation of at least a part of said computation task. This may provide the advantage that the assignment of boosters to computation nodes can be performed dynamically at runtime.

[0031] The object is also solved by a method for operating a computer cluster arrangement according to the features of PCT patent claim 11 (presented below).

[0032] Accordingly a method for operating a computer cluster arrangement is provided for processing a computation task, the method comprising:

[0033] computing at least a first part of the computation task by at least two of the plurality of computation nodes, each computation node interfacing a communication infrastructure;

[0034] computing at least a second part of the computation task by at least one booster, each booster interfacing the communication infrastructure; and

[0035] assigning at least one booster to one of the plurality of computation nodes by a resource manager for computation of the second part of the computation task, the assignment being accomplished as a function of a predetermined assignment metric.

[0036] Furthermore, a computer program being configured for accomplishing the introduced method as well as a computer readable medium for storing the computer program product are provided.

[0037] Preferred embodiments of the invention are defined in the following claims of the Published PCT Application:

[0038] 1. A computer cluster arrangement for processing a computation task, comprising:

[0039] a plurality of computation nodes (CN), each of which interfacing a communication infrastructure (IN), at least two of which being arranged to jointly compute at least a first part of said computation task;

[0040] at least one booster (B) being arranged to compute at least a second part of said computation task, each booster (B) interfacing said communication infrastructure (IN); and

[0041] a resource manager (RM) being arranged to assign at least one booster (B) to at least one of said plurality of computation nodes (CN) for computation of said second part of said computation task, the assignment being accomplished as a function of a predetermined assignment metric.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 162 of 411

[0042] 2. The computer cluster arrangement according to claim 1, wherein said predetermined assignment metric is formed according to at least one of a group of metric specification techniques, said group comprising: a temporal logic, an assignment matrix, an assignment table, a probability function and a cost function.

[0043] 3. The computer cluster arrangement according to one of the preceding claims, wherein said predetermined assignment metric is specified as a function of at least one of a group of assignment parameters, said group comprising: resource information, cost information, complexity information, scalability information, a computation log record, compiler information, priority information and a time stamp.

[0044] 4. The computer cluster arrangement according to one of the preceding claims, wherein said assignment of at least one booster (B) to one of said plurality of computation nodes (CN) triggers at least one of a group of signals, said group comprising: a remote procedure call, a parameter handover, a data transmission.

[0045] 5. The computer cluster arrangement according to one of the preceding claims, wherein each computation node (CN) and each booster (B) interface said communication infrastructure (IN) respectively via an interfacing unit (IU**1**; IU**2**).

[0046] 6. The computer cluster arrangement according to claim 5, wherein said interfacing unit (IU**1**; IU**2**) comprises at least one of a group of components, said group comprising: a virtual interface, a stub, a socket, a network controller and a network device.

[0047] 7. The computer cluster arrangement according to one of the preceding claims, wherein said communication infrastructure (IN) comprises at least one of a group of components, said group comprising: a bus, a communication link, a switching unit, a router and a high speed network.

[0048] 8. The computer cluster arrangement according to one of the preceding claims, wherein each computation node (CN) comprises at least one of a group of components, said group comprising: a multi-core processor, a cluster, a computer, a workstation and a multi-purpose processor.

[0049] 9. The computer cluster arrangement according to one of the preceding claims, wherein said at least one booster (B) comprises at least one of a group of components, said group comprising: a many-core processor, a scalar processor, a co-processor, a graphical processing unit, a cluster of many-core processors and a monolithic processor.

[0050] 10. The computer cluster arrangement according to one of the preceding claims, wherein said resource manager (RM) is arranged to update said predetermined assignment metric during computation of at least a part of said computation task.

[0051] 11. A method for operating a computer cluster arrangement for processing a computation task, especially a computer cluster according to at least one of claims 1 to 10, comprising:

[0052] computing (**100**) at least a first part of said computation task by at least two of said plurality of computation nodes (CN), each computation node (CN) interfacing a communication infrastructure (IN);

[0053] computing (**101**) at least a second part of said computation task by at least one booster (B), each booster (B) interfacing said communication infrastructure (IN); and

[0054] assigning (**102**) at least one booster (B) to one of said plurality of computation nodes (CN) by a resource manager (RM), for computation of said second part of said computation task, said assignment being accomplished as a function of a predetermined assignment metric.

[0055] 12. A computer program product being configured for accomplishing the method according to claim 11.

[0056] 13. A computer readable medium for storing said computer program product according to claim 12.

BRIEF DESCRIPTION OF THE DRAWINGS

[0057] The invention will now be described merely by way of illustration with reference to the accompanying drawings:

[0058] FIG. **1** shows a computer cluster arrangement according to the state of the art.

[0059] FIG. **2** shows a schematic illustration of a computer cluster arrangement according to an aspect of the present invention.

[0060] FIG. **3** shows a schematic illustration of a computer cluster arrangement according to a further aspect of the present invention.

[0061] FIG. **4** shows a schematic illustration of a method for operating a computer cluster arrangement according to an aspect of the present invention.

[0062] FIG. **5** shows a schematic illustration of a method for operating a computer cluster arrangement according to a further aspect of the present invention.

[0063] FIG. **6** shows a schematic illustration of control flow of a computer cluster arrangement according to a further aspect of the present invention.

[0064] FIG. **7** shows a schematic illustration of control flow implementing reverse acceleration of a computer cluster arrangement according to a further aspect of the present invention.

[0065] FIG. **8** shows a schematic illustration of control flow of a computer cluster arrangement according to a further aspect of the present invention.

[0066] FIG. **9** shows a schematic illustration of network topology of a computer cluster arrangement according to an aspect of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

[0067] In the following same notions will be denoted with the same reference signs, if not indicated otherwise. FIG. **2** shows a computer cluster arrangement comprising a cluster C as well as a booster group BG. The cluster comprises in the present embodiment four computation nodes, also referred as CN, as well as three boosters, also referred to as B. A flexible coupling of boosters to computation nodes is established by a communication infrastructure IN, such as a so called interconnect. This kind of communication infrastructure IN can be implemented for instance by using InfiniBand. Hence, each of the boosters B can be shared by any of the computation nodes CN. Furthermore a virtualization on cluster level can be accomplished. Each booster, or at least a part of the boosters, can be virtualized and made available to the computation nodes virtually.

[0068] In the present embodiment computation tasks are processed by at least one of the computation nodes CN and at least a part of the computation tasks may be forwarded to at least one of the boosters B. The boosters B are arranged to compute specific problems and provide specific processing power. Hence, problems can be outsourced from one of the computation nodes CN to the boosters B, be computed by the booster and the result may be delivered back to the computation node. The assignment of boosters ESB to computation

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 163 of 411

nodes CN can be accomplished by a resource manager, also referred to as RM. The resource manager initializes a first assignment and further on establishes a dynamic assignment of boosters B to computation nodes CN.

[0069] For communication between boosters and computation nodes an application programming interface, also referred to as API, can be provided. The boosters B may be controlled transparently by the computation nodes through the respective API function calls. The API abstracts and enhances actual native programming models of the boosters. Furthermore the API may provide means for fault tolerance in case of a booster failure. A communication protocol involved in API calls may be layered on top of a communication layer. In the following a short description of a set of API calls according to an aspect of the present invention is provided, in which the parameter "accelerator" may specify the addressed booster:

aanInit (accelerator)

[0070] Initializes the booster before use

aanFinalize (accelerator)

[0071] Releases bookkeeping information on the booster after use

aanMemAlloc (address, size, accelerator)

[0072] Allocates size Bytes of memory on the referenced booster

[0073] Returns address of allocated device memory

aanMemFree (address, accelerator)

[0074] Releases the memory starting at address on the referenced booster

aanMemCpy (dst, src, size, direction, accelerator)

[0075] Copies size Bytes from src to dst memory address

[0076] The direction of the copy operation can be:

[0077] (i) booster to host,

[0078] (ii) host to booster

aanKernelCreate (file_name, funct_name, kernel, accelerator)

[0079] Creates a kernel defined by the name of the file (file_name) and the name of the function (funct_name) for execution on the referenced booster

[0080] Returns handle to kernel

aanKernelSetArg (kernel, index, size, align, value)

[0081] Defines argument for kernel execution by its index in the argument list, size, alignment requirement (align), and value

aanKernelRun (kernel, grid_dim, block dim)

[0082] Starts kernel execution on the booster associated with kernel in a previous call to acKernelCreate( ) The number of threads is determined by number of threads per block (block_dim) and number of blocks in the grid (grid_dim)

aanKernelFree (kernel)

[0083] Releases the resources associated with kernel

[0084] FIG. **3** shows a further cluster arrangement according to an aspect of the present invention. The depicted computer cluster arrangement is arranged to compute scientific computation tasks, especially in the context of high performance cluster technology. A closer analysis of the characteristics of the portfolio of scientific high performance cluster application codes reveals that many codes with Exascale needs include, on the one hand, code blocks that are well suited for Exascaling, and, on the other hand, such code blocks that are too complex to be so scalable. In the following, the distinction between highly scalable and complex is made on the level of code blocks, and we introduce the notions Exascale Code Blocks (ECB) and complex Code Blocks (CCB). Obviously, there is no purely highly scalable code, and there is no strictly complex code as well. Each code has highly scalable and less scalable complex elements. In fact, there is a continuum between both extremes. Interestingly, many less scalable elements of a code do not require high scalability but instead require large local memory. It is also evident that all-to-all communication elements have a high advantage under smaller parallelism.

[0085] For such problems, where a decent balance between ECBs and CCBs is given in terms of the relative amounts of memory (i.e., the degrees of freedom handled in of the relative amounts of memory, i.e., the degrees of freedom handled in of ECB vs. the CCB), execution times and data to be exchanged, it suggests itself to adapt to this situation by means of a specific architectural solution. The solution consisting of a traditional cluster computer approach along with an Exascale booster with tightly connected boosters and being connected with a cluster through the cluster's network. This dualistic approach has the potential to widen the anticipated narrow application field of pure Exascale systems substantially.

[0086] A coarse-grained architectural model emerges, where the highly scalable parts or ECBs of an application code are executed on a parallel many-core architecture, which is accessed dynamically, while the CCBs are executed on a traditional cluster system suitable dimensioned, including the connectivity along with a refined dynamical resource allocation system.

[0087] Clusters at Exascale require virtualization elements in order to guarantee resilience and reliability. While local accelerators, in principle, allow for a simple view on the entire system and in particular can utilize the extremely high local bandwidth, they are absolutely static hardware elements, well suited for farming or master-slave parallelization. Hence, it would be difficult to include them in a virtualization software layer. In addition, there would be no fault tolerance if an accelerator fails, and there was no tolerance for over or under subscription.

[0088] The cluster's computation nodes CN are internally coupled by a standard cluster interconnect, e.g., Mellanox InfiniBand. This network is extended to include the booster (ESB) as well. In the figure we have drawn three such boosters. The ESBs each consist of a multitude of many-core accelerators connected by a specific fast low-latency network. This connection of the CNs with the ESBs is very flexible. A sharing of accelerator capability between computation nodes becomes possible. The virtualization on the cluster level is not hampered by the model and the full ESB parallelism can be exploited. The ESB-to-CN assignment proceeds via a dynamical resource manager RM. A static assignment at start-time can be made dynamic at run-time. All CN-ESB communication proceeds via the cluster network protocol. The intra-AC communication will require new solutions. The ESB allocation can follow the application needs and fault tolerance is guaranteed in case of accelerator failures while all computation nodes share the same growth capacity.

[0089] As compute element of the booster Intel's many-core processor Knight's Corner (KC) may be applied. The KC-chip will consist of more than 50 cores and is expected to provide a DP compute capacity of over 1 Teraflop/s per chip. With 10,000 elements a total performance of 10 Petaflop/s would be in reach. The predecessor of KC, the Knight's Ferry

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 164 of 411

processor (KF) will be used in the project to create a PCIe-based pilot system to study the cluster-booster (CN-ESB) concept.

[0090] As the compute speed of KF exceeds current commodity processors by a factor of about 10, the intra-ESB communication system has to be dimensioned accordingly. The ESB's communication system requires at least 1 Terabit/s per card (duplex). The communication system EXTOLL may be used as an implementation of a bus system, which provides a communication rate of 1.44 Terabit/s per card. It realizes a 3d topology providing 6 links per card. Concerning its simplicity, this topology appears to be applicable for a booster based on many-core accelerators. Even with two directions reserved for cut-through routing, EXTOLL can saturate the PCI Express performance as far as the data rate is concerned. The latency can reach 0.3 μs, when based on an ASIC realization. Currently, EXTOLL is realized by means of FPGAs.

[0091] FIG. **4** shows a flow diagram for illustrating an aspect of a method for operating a computer cluster arrangement according to the present invention. In a first step **100** at least the first part of a computation task is computed by at least two of the plurality of computation nodes CN, each computation node CN interfacing a communication infrastructure IN. Furthermore, computing of at least a second part of the computation task in step **101** by at least one booster B is performed, each booster B interfacing the communication infrastructure IN. Further, assigning at least one booster B to one of the plurality of computation nodes CN in step **102** by a resource manager RM, for computation of the second part of the computation task is performed. As the right arrow in FIG. **4** indicates the control flow may point back to step **100**. After assigning at least one booster B to at least one of the plurality of computation nodes CN in step **102** the assignment can be communicated to a computation node CN, which uses the transmitted assignment in further outsourcing steps. Hence, computing at least a second part of the computation task is performed in step **101** as a function of the assignment step **102**.

[0092] FIG. **5** shows a flow diagram illustrating a method for operating a computer cluster arrangement according to an aspect of the present invention. In the present embodiment after the assignment in step **202** of the at least one booster B to one of the plurality of computations nodes CN the step of computing **201** at least a second part of the computation task is performed. Hence, it is possible to select a specific booster B and based on the assignment being established in step **202** a booster B computes the at least second part of the computation task. This may be of advantage in case the at least second part of the computation task is forwarded to the resource manager RM, which assigns a booster B to the second part of the computation task. The resource manager RM can then transmit the second part of the computation task to the booster B, without the necessity that the computation node CN directly contacts the booster B.

[0093] Referring to FIGS. **4** and **5** the person skilled in the art appreciates that any of the steps can be performed iteratively, in a different order and may comprise further sub steps. For instance step **102** may be performed before step **101**, which results in a computation of a first part of the computation task, an assignment of one booster to one computation node and finally computation of the second part of the computation task. Step **102** may comprise sub steps such as returning the computed at least second part of the computation task back to the computation node CN. Hence, the booster B returns the computed result back to the computation nodes CN. The computation nodes CN may use the returned value for computation of further computation tasks and may again forward at least a further part of a computation task to at least one of the boosters B.

[0094] FIG. **6** shows a block diagram of control flow of a computer cluster arrangement according to an aspect of the present invention. In the present embodiment a computation node CN receives a computation task and requests a booster B for outsourcing at least a part of the received computation task. Therefore, a resource manager RM is accessed, which forwards the part of the computation task to a selected booster B. The booster B computes the part of the computation task and returns a result, which is indicated by the most right arrow. According to a further aspect of the present embodiment the return value can be passed back to the computation node CN.

[0095] FIG. **7** shows a block diagram of control flow, implementing reverse acceleration, of a computer cluster arrangement according to an aspect of the present invention. In the present embodiment an acceleration of computation of computation tasks being computed by at least one booster B is performed by assigning at least one computation node CN to at least one booster B. Hence the control and information flow is reversed as regards the embodiment being shown in FIG. **6**. Computation of tasks can therefore be accelerated by outsourcing computation tasks from the boosters B to at least one computation node CN.

[0096] FIG. **8** shows a block diagram of control flow of a computer cluster arrangement according to a further aspect of the present invention. In the present embodiment the resource manager RM does not pass the at least one part of the computation task to the booster B, but the computation node CN requests an address or a further identification of a booster B, which is arranged to compute the specific at least one part of the computation task. The resource manager RM returns the required address to the computation node CN. The computation node CN is now able to directly access the booster B by means of the communication infrastructure IN. In the present embodiment the communication infrastructure IN is accessed via interfacing units. The computation nodes CN accesses the communication infrastructure IN by interfacing unit IU**1** and the booster B interfaces the communication infrastructure IN by interfacing unit IU**2**.

[0097] Furthermore, the resource manager RM is arranged to evaluate the resource capacities of the booster B and performs the assignment, which means the selection of the booster B, as a function of the evaluated resource capacities of each of the boosters B. For doing so the resource manager RM may access the assignment metric, which may be stored in a database DB or any kind of data source. The resource manager RM is arranged to update the assignment metric, which can be performed under usage of a database management system. The database DB can be implemented as any kind of storage. It may for instance be implemented as a table, a register or a cache.

[0098] FIG. **9** shows a schematic illustration of network topology of a computer cluster arrangement according to an aspect of the present invention.

[0099] In one embodiment the computation nodes share a common, first, communication infrastructure, for instance a star topology with a central switching unit S. A further, second, communication infrastructure is provided for communi-

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 165 of 411

cation of the computation nodes CN with booster nodes BN. A third communication infrastructure is provided for communication among booster nodes BN. Hence, a high speed network interface for communication among booster nodes BN can be provided with a specific BN-BN communication interface. The BN-BN communication infrastructure can be implemented as a 3d topology.

[0100] In a further embodiment two communication infrastructures are provided, one for communication among computation nodes CN and one further communication infrastructure for communication among booster nodes BN. Both communication infrastructures can be coupled by at least one communication link from the first network to the second network or from the second network to the first network. Hence, one selected computation node CN or one selected booster node BN is connected with the respectively other network. In the present FIG. **9** one booster node BN is connected with the communication infrastructure of the computation nodes CN under usage of a switching unit S.

[0101] In a further embodiment the booster group BG itself may be connected to the communication infrastructure of the computation nodes CN or an intermediate communication infrastructure. The communication infrastructures may generally differ among other characteristics in their topology, bandwidth, communication protocols, throughput and message exchange. A booster B may for example comprise 1 to 10.000 booster nodes BN, but is not restricted to this range. The resource manager RM may generally manage parts of the booster nodes BN and can therefore partition the overall number of booster nodes BN and dynamically form boosters B out of said number of booster nodes BN. The switching unit S may be implemented by a switch, a router or any network device.

[0102] The person skilled in the art appreciates further arrangements of the components of the computer cluster arrangement. For instance the database DB may be accessed by further components, respectively nodes of the computer cluster arrangement. The illustrated computation nodes CN as well as the illustrated booster group BG may be one of many further computation nodes CN as well as one of many booster groups BG, respectively, which access the resource manager RM and/or the communication infrastructure IN. Furthermore acceleration can also be performed reversely by outsourcing at least a part of the computation task from at least one booster B to at least one computation node.

What is claimed is:

1. A computer cluster arrangement for processing a computation task, comprising:

a plurality of computation nodes (CN), each of which interfacing a communication infrastructure (IN), at least two of which being arranged to jointly compute at least a first part of said computation task;

at least one booster (B) being arranged to compute at least a second part of said computation task, each booster (B) interfacing said communication infrastructure (IN); and

a resource manager (RM) being arranged to assign at least one booster (B) to at least one of said plurality of computation nodes (CN) for computation of said second part of said computation task, the assignment being accomplished as a function of a predetermined assignment metric.

2. The computer cluster arrangement according to claim **1**, wherein said predetermined assignment metric is formed according to at least one of a group of metric specification techniques, said group comprising: a temporal logic, an assignment matrix, an assignment table, a probability function and a cost function.

3. The computer cluster arrangement according to claim **1**, wherein said predetermined assignment metric is specified as a function of at least one of a group of assignment parameters, said group comprising: resource information, cost information, complexity information, scalability information, a computation log record, compiler information, priority information and a time stamp.

4. The computer cluster arrangement according to claim **2**, wherein said predetermined assignment metric is specified as a function of at least one of a group of assignment parameters, said group comprising: resource information, cost information, complexity information, scalability information, a computation log record, compiler information, priority information and a time stamp.

5. The computer cluster arrangement according to claim **1**, wherein said assignment of at least one booster (B) to one of said plurality of computation nodes (CN) triggers at least one of a group of signals, said group comprising: a remote procedure call, a parameter handover, a data transmission.

6. The computer cluster arrangement according to claim **2**, wherein said assignment of at least one booster (B) to one of said plurality of computation nodes (CN) triggers at least one of a group of signals, said group comprising: a remote procedure call, a parameter handover, a data transmission.

7. The computer cluster arrangement according to claim **3**, wherein said assignment of at least one booster (B) to one of said plurality of computation nodes (CN) triggers at least one of a group of signals, said group comprising: a remote procedure call, a parameter handover, a data transmission.

8. The computer cluster arrangement according to claim **4**, wherein each computation node (CN) and each booster (B) interface said communication infrastructure (IN) respectively via an interfacing unit (IU**1**; IU**2**).

9. The computer cluster arrangement according to claim **5**, wherein said interfacing unit (IU**1**; IU**2**) comprises at least one of a group of components, said group comprising: a virtual interface, a stub, a socket, a network controller and a network device.

10. The computer cluster arrangement according to claim **1**, wherein said communication infrastructure (IN) comprises at least one of a group of components, said group comprising: a bus, a communication link, a switching unit, a router and a high speed network.

11. The computer cluster arrangement according to claim **1**, wherein each computation node (CN) comprises at least one of a group of components, said group comprising: a multi-core processor, a cluster, a computer, a workstation and a multi-purpose processor.

12. The computer cluster arrangement according to claim **1**, wherein said at least one booster (B) comprises at least one of a group of components, said group comprising: a many-core processor, a scalar processor, a co-processor, a graphical processing unit, a cluster of many-core processors and a monolithic processor.

13. The computer cluster arrangement according to claim **1**, wherein said resource manager (RM) is arranged to update said predetermined assignment metric during computation of at least a part of said computation task.

14. A method for operating a computer cluster arrangement for processing a computation task, said computer cluster

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 166 of 411

being selected from the computer clusters according to at least one of the preceding claims, comprising:

computing (**100**) at least a first part of said computation task by at least two of said plurality of computation nodes (CN), each computation node (CN) interfacing a communication infrastructure (IN);

computing (**101**) at least a second part of said computation task by at least one booster (B), each booster (B) interfacing said communication infrastructure (IN); and

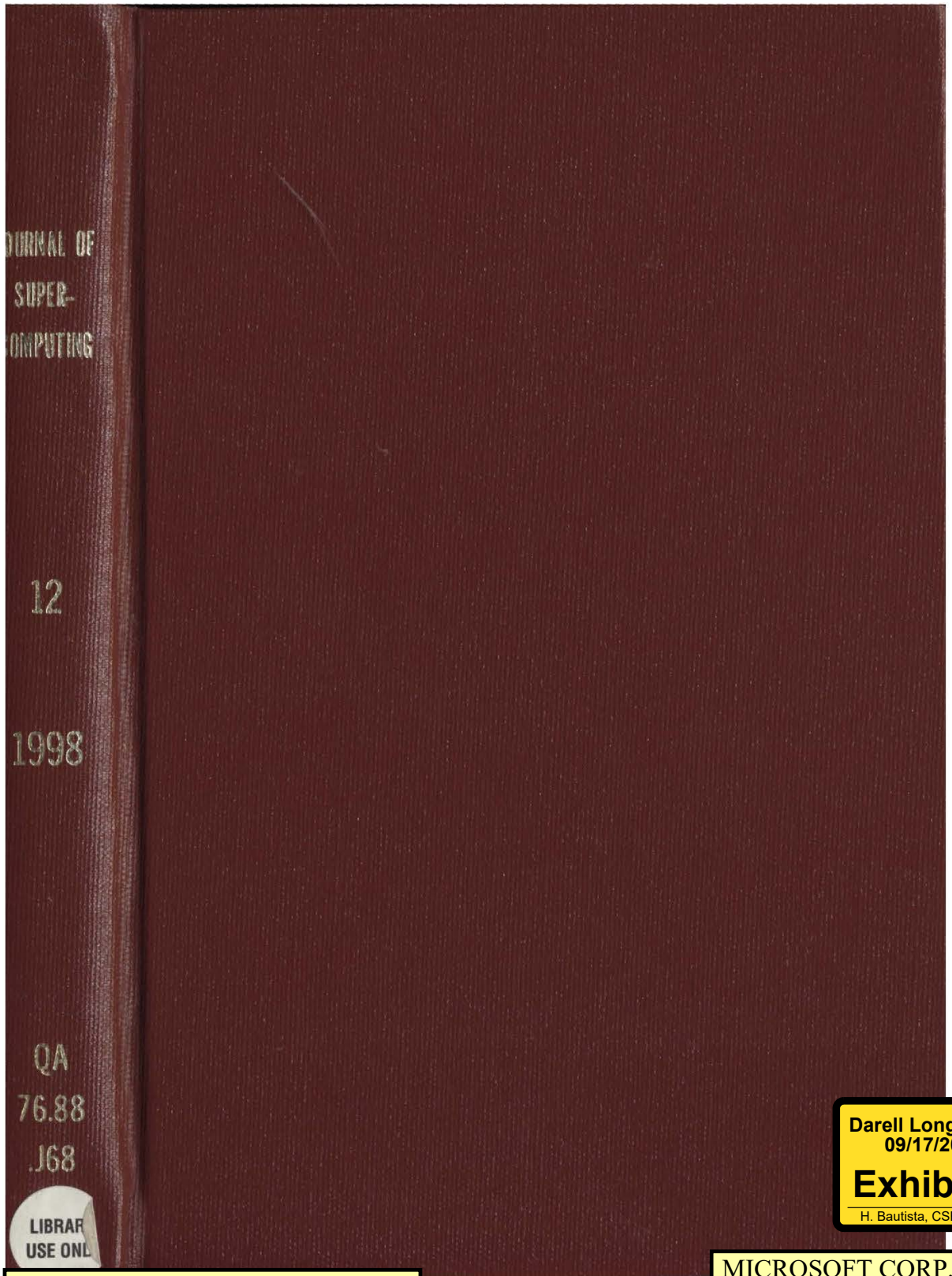
assigning (**102**) at least one booster (B) to one of said plurality of computation nodes (CN) by a resource manager (RM), for computation of said second part of said computation task, said assignment being accomplished as a function of a predetermined assignment metric.

15. A computer program product being configured for accomplishing the method according to claim **14**.

16. A computer readable medium for storing said computer program product according to claim **15**.

* * * * *

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 167 of 411

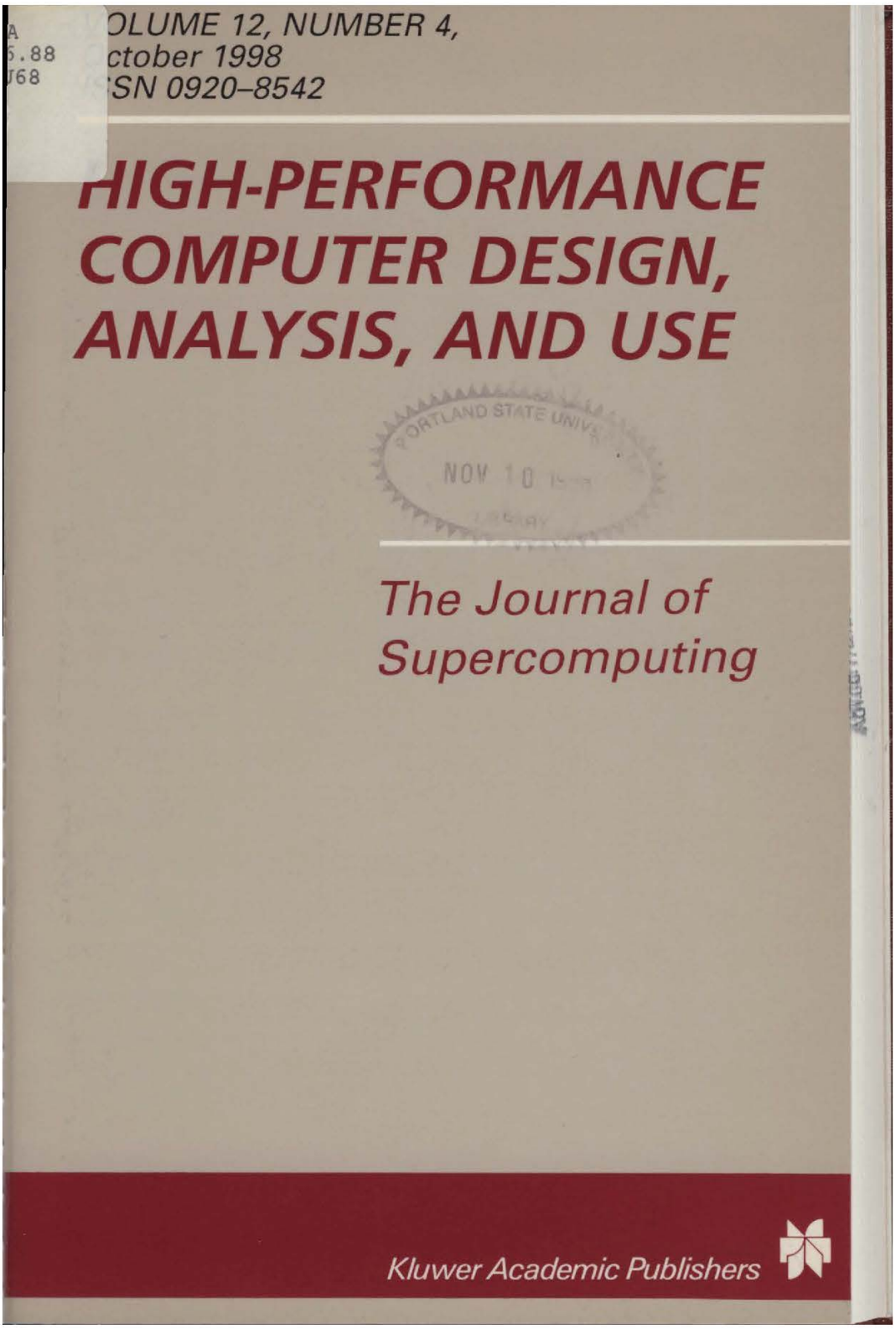


Darell Long, Ph.D.
09/17/2025
Exhibit D
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1005

Petitioner has added romanettes to pages (i)-(vi). Otherwise, it leaves the original page numbering.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 168 of 411



ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 169 of 411

PORTLAND STATE UNIVERSITY Library

THE JOURNAL OF SUPERCOMPUTING

Editor-in-Chief

Hamid Arabnia, University of Georgia

Subject Area Editors

APPLICATIONS: David H. Bailey, NASA Ames Research Center
ALGORITHMS: Kenneth Neves, Boeing Computer Services
PERFORMANCE: Joanne Martin, IBM Corporation
ARCHITECTURE AND SYSTEMS: George Michael, Lawrence Livermore National Laboratory

ISSN: 0920–8542

© 1998 Kluwer Academic Publishers, Boston/U.S.A.

No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

Manufactured in the Netherlands

Board of Review

Makoto Amamiya, *Kyushu University*
Arvind, *Massachusetts Institute of Technology*
William F. Ballhaus, Jr., *Martin Marietta Astronautics*
David Barkai, *NASA Ames Research Center*
Francine Berman, *University of California at San Diego*
Alfred Brenner, *Institute for Defense Analysis*
Ernest Brickell, *CertCo*
Harvey Cragon, *The University of Texas at Austin*
Narsingh Deo, *University of Central Florida*
Jack J. Dongarra, *University of Tennessee*
Geoffrey C. Fox, *Syracuse University*
John Gregory, *Westinghouse Electric Corporation*
Milton Halem, *NASA Goddard Space Flight Center*
Harry Jordan, *University of Colorado*
Sidney Karin, *GA Technologies*
David Klappholz, *Stevens Institute of Technology*
Gordon Lyon, *National Institute of Standards and Technology*
Kenichi Miura, *Fujitsu America*
Ronald C. Mullin, *University of Waterloo*
Carl Pomerance, *The University of Georgia*
Paul B. Schneck, *MRJ Technology Solutions*
Jacob T. Schwartz, *Courant Institute of Mathematical Science*
Daniel Siewiorek, *Carnegie-Mellon University*
Burton J. Smith, *TERA Computer Company*
James E. Smith, *University of Wisconsin - Madison*
Lloyd Thorndyke, *Consultant*
Jack Worlton, *Worlton and Associates*
Stephen S.Yau, *University of Florida*

Second Class Postage Paid Rahway, N.J., Mercury Airfreight International Ltd. Inc., 2323 Randolph Avenue, Avenel, N.J. 07001
ISSN: 0920–8542 USPS: 003–163
Postmaster: Please send all address corrections to: *The Journal of Supercomputing,* c/o Mercury Airfreight International Ltd. Inc., 2323 Randolph Avenue, Avenel, N.J. 07001, USA.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 170 of 411

VOLUME 12, NUMBER 4, OCTOBER 1998

THE JOURNAL OF SUPERCOMPUTING

CONTENTS

Distribution-Independent Hierarchical Algorithms for the N-Body Problem
Srinivas Aluru, John Gustafson, G.M. Prabhu and Fatih E. Sevilgen 303

A Robust Compile Time Method for Scheduling Task Parallelism on Distributed Memory Machines
Sekhar Darbha and Santosh Pande 325

Compiler Support for Array Distribution on NUMA Shared Memory Multiprocessors
Tarek S. Abdelrahman and Thomas N. Wong 349

An Efficient Systolic Algorithm for the Longest Common Subsequence Problem
Yen-Chun Lin and Jyh-Chian Chen 373

A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms
John R. Budenske, Ranga S. Ramanujan and Howard Jay Siegel 387

The Journal of Supercomputing is abstracted and/or indexed in *ISI: Current Contents: Engineering, Technology and Applied Science, ISI: Science Citation Index, ISI: Scisearch, ISI: Research Alert, ISI: Compumath Citation Index, INSPEC Information Services, Engineering Index, COMPENDEX* Plus database, Computer Literature Index, Artificial Intelligence Abstracts, Computer and Information Systems Abstracts, Information Science Abstracts, ACM Guide to Computing Literature, ACM Computing Reviews, Computer Abstracts, ISI: Automatic Subject Citation Alert, Zentralblatt fur Mathematik, Referativnyi Zhurnal, MATH (Database), CompactMATH (CD-ROM), Compuscience (Database), PASCAL Database,* and *Ei Page One.*

0920-8542(199810)12:4;1-I

CODEN JOSUED ISSN 0920-8542

PORTLAND STATE UNIVERSITY

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 171 of 411

The Journal of Supercomputing

Volume 12, No. 4, October 1998

Distribution-Independent Hierarchical Algorithms for the N-Body Problem ..... *Srinivas Aluru, John Gustafson, G.M. Prabhu and Fatih E. Sevilgen* 303

A Robust Compile Time Method for Scheduling Task Parallelism on Distributed Memory Machines ..... *Sekhar Darbha and Santosh Pande* 325

Compiler Support for Array Distribution on NUMA Shared Memory Multiprocessors ..... *Tarek S. Abdelrahman and Thomas N. Wong* 349

An Efficient Systolic Algorithm for the Longest Common Subsequence Problem ..... *Yen-Chun Lin and Jyh-Chian Chen* 373

A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms ..... *John R. Budenske, Ranga S. Ramanujan and Howard Jay Siegel* 387

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 172 of 411

The Journal of Supercomputing is abstracted and/or indexed in ***ISI: Current Contents: Engineering, Technology and Applied Science, ISI: Science Citation Index, ISI: Scisearch, ISI: Research Alert, ISI: Compumath Citation Index, INSPEC Information Services, Engineering Index, COMPENDEX Plus database, Computer Literature Index, Artificial Intelligence Abstracts, Computer and Information Systems Abstracts, Information Science Abstracts, ACM Guide to Computing Literature, ACM Computing Reviews, Computer Abstracts, Applied Mechanics Reviews, ISI: Automatic Subject Citation-Alert, Zentralblatt für Mathematik, Referativny: Zhurnal, MATH (Database), CompactMATH (CD-ROM), Compuscience (Database), PASCAL Database, Ei Page One.***

Subscription Rates

The subscription price for Volume 12 (4 quarterly issues), including postage and handling, is:
For institutions: US$349.00 Dfl. 680,—
For individuals: US$175.00 Dfl. 320,—

Ordering Information/Sample Copies

Subscription orders and requests for sample copies should be sent to:

Kluwer Academic Publishers
Post Office Box 358
Accord Station
Hingham, MA 02018-0358, USA

or

Kluwer Academic Publishers
Post Office Box 322
3300 AH Dordrecht
THE NETHERLANDS

Subscriptions may also be sent to any subscription agent.

Printed on acid-free paper.

Photocopying. *In the U.S.A.*: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by Kluwer Academic Publishers for users registered with the Copyright Clearance Center (CCC) Transactional Reporting Service, provided that the fee of $9.50 per copy per article is paid directly to CCC. For those organizations that have been granted a photocopy license by CCC, a separate system of payment has been arranged. The fee code for users of the Transactional Reporting Service is 0920-8542/98/$9.50.

Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising purposes or promotional purposes, for creating new collective works, or for resale.

In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to Kluwer Academic Publishers, P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

ISSN 0920-8542

© 1998 Kluwer Academic Publishers. Manufactured in The Netherlands.
No part of the material protected by this copyright notice may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without written permission from the copyright owner.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 173 of 411

The Journal of Supercomputing, 12:387–406 (1998)
© 1998 Kluwer Academic Publishers, Boston. Manufactured in the Netherlands

A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms

JOHN R. BUDENSKE AND RANGA S. RAMANUJAN
Architecture Technology Corporation, P.O. Box 24344, Minneapolis, MN 55424, USA, {budenske, ranga}@atcorp.com

HOWARD JAY SIEGEL
Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA, hj@purdue.edu

Abstract. This study focuses on a particular application domain (iterative automatic target recognition tasks) and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, a methodology is presented for the on-line operating system to decide heuristically whether to perform a remapping of the application onto the platform based on information generated from input data by the application during execution. If the decision is to remap, the operating system will be able to select a mapping, which is appropriate for the given state of the application, from a stored set of mappings that were previously derived with an off-line heuristic.

Keywords: Algorithm mapping, automatic target recognition, genetic algorithms, heterogeneous computing, real-time processing

1. Introduction

Heterogeneous computing (*HC*) covers a great variety of situations (e.g., see [9], [17], [22], [23], [26]). This study focuses on a particular application domain (iterative automatic target recognition (*ATR*) tasks) and an associated specific class of dedicated heterogeneous parallel hardware platforms. The contribution of this paper is that, for the computational environment considered, it presents a methodology for real-time on-line input-data dependent remapping of the application subtasks to the processors in the heterogeneous parallel hardware platform using a previously stored off-line statically determined *mapping* (i.e., a matching of application subtasks to processors and a scheduling for the execution order of these subtasks). That is, the operating system will be able to make a heuristically-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system will be able to select a previously heuristically derived and stored mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking).

This high-level operating system approach, which includes the capability for the on-line use of off-line computed mappings, is called the *IOS* (Intelligent Operating System). The IOS conceptual design, off-line components, and on-line components are depicted in Figures 1, 2, and 3, respectively. Consider the conceptual design (Figure 1). The *ATR Kernel* makes decisions on how a given ATR application task should be accomplished, including determining the partial ordering of subtasks and which algorithms should be used to accomplish each subtask. The *HC Kernel* makes a heuristically-based decision about how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. Also, the HC Kernel interacts with the *Basic Kernel* (the low-level operating system) to execute the application and monitor its execution so it can decide when to remap the subtasks onto the hardware platform. Thus, the ATR Kernel deals with application issues, while the HC Kernel deals with implementation issues. Information from the *Algorithm Database* and the *Knowledge Base* is used to support the ATR and HC Kernels. This design has its roots in the high-level model presented in [4] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors (e.g. PASM).

This paper concentrates on the operation of the HC Kernel; the other components will be discussed only to the extent needed for describing the HC Kernel (additional information on the other components is in [2], [3]). In particular, this paper focuses on (1) the

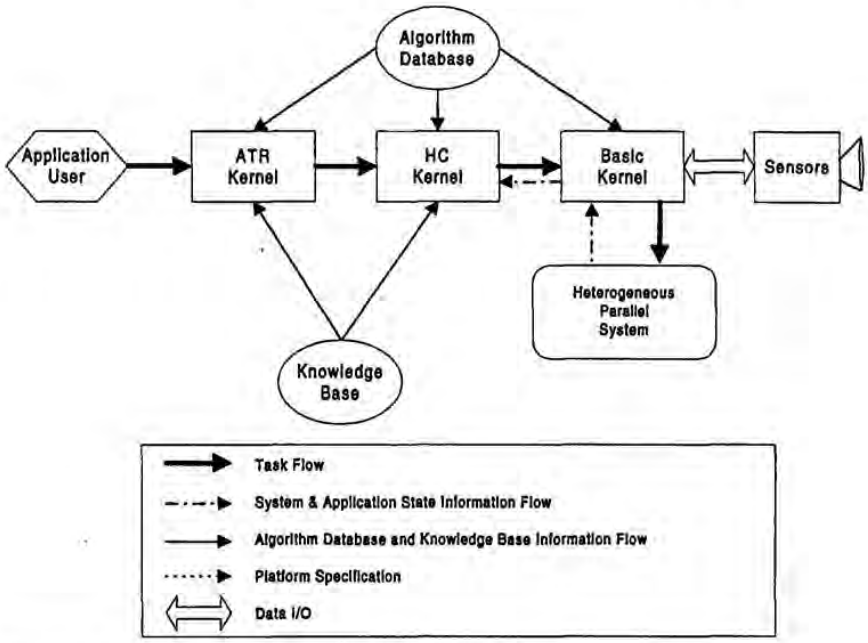


Figure 1. IOS Conceptual Design

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 175 of 411

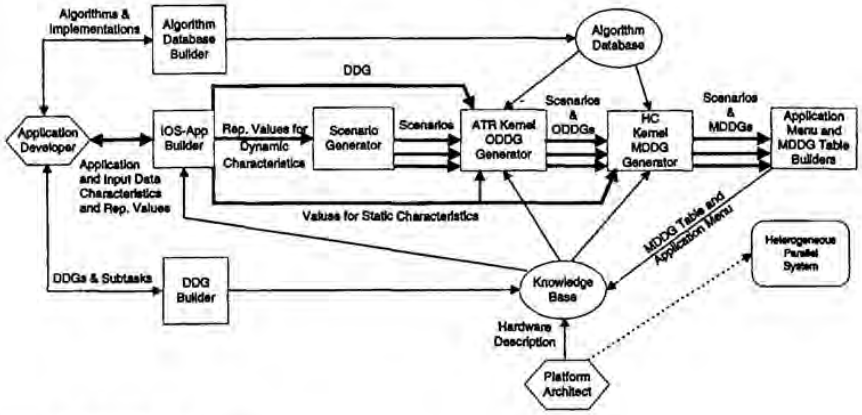


Figure 2. IOS Off-line Components

application and hardware platform characteristics that are needed to enable the use of the HC Kernel, (2) the techniques that comprise the HC Kernel, and (3) how the information needed for the HC Kernel to operate is collected. The IOS has not been implemented; such an implementation is a major undertaking and outside the scope of this paper, which is the design concepts for the HC Kernel.

The HC Kernel differs from other real-time HC mapping techniques in that it allows on-line real-time use of off-line precomputed mappings. This is significant because it is possible for off-line heuristics to produce better mappings than on-line heuristics. This is a result of the fact that off-line heuristics can have much longer execution times to search

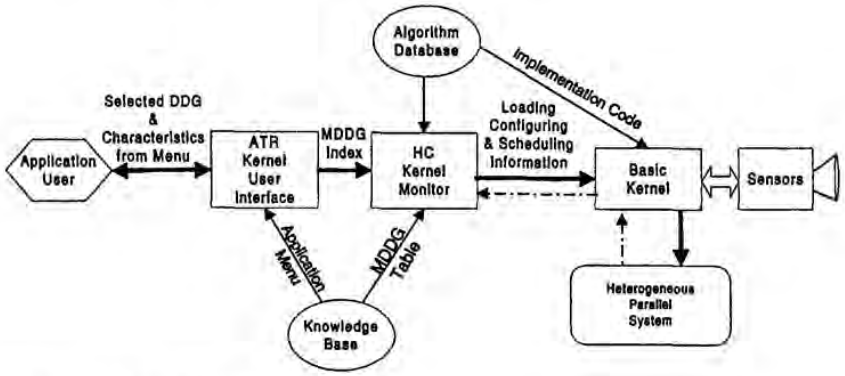


Figure 3. IOS On-line Components

for a good solution than what is practical for an on-line heuristic. Thus, with the HC Kernel, the mapping quality of an off-line time-consuming heuristic can be approached at real-time speeds.

The HC Kernel ideas can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the iterative ATR applications and platforms considered here. Examples of other such application domains are sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection.

Thus, the goal of this paper is to introduce the ideas underlying the design and use of the HC Kernel for a particular class of application computational structure and a particular type of target hardware platform. The performance of the HC Kernel in comparison to other methods will depend upon the exact application and exact platform under consideration. The technique presented here is offered to future system designers to evaluate as an alternative for their particular situations.

The paper is organized as follows. Section 2 describes the application domain and Section 3 the heterogeneous parallel hardware platform. Overviews of the off-line and on-line components of the IOS are presented in Sections 4 and 5, respectively. More information about the HC Kernel off-line and on-line components are provided in Sections 6 and 7, respectively.

2. Application domain

Simply stated, an ATR system takes a set of images from a group of sensors and produces some type of description of the scene. The various types of processing required in an ATR system can be broadly classified into three groups: numeric computation, quasi-symbolic computation (e.g., where numeric and symbolic types of operations are used to describe surfaces and shapes of objects in the scene), and symbolic computation (e.g., used to produce the scene description). Heterogeneous parallel architectures are ideal computing platforms for efficiently handling computational tasks with such diverse requirements.

A key technical issue that must be addressed to exploit the inherent potential of heterogeneous parallel computing systems to efficiently implement ATR applications is the development of a high-level operating system that can fully utilize the architectural flexibility of such a system. Such a high-level operating system must be able to assign each ATR application subtask to the processors where it is best suited for execution. Often, subtasks can execute concurrently, sharing resources. Because the execution time of application subtasks in an ATR system is highly input-data dependent (e.g., number of currently located objects), this matching and scheduling of application subtasks to processors must be performed dynamically at run time.

This work is being developed for a class of ATR applications each of which can be modeled as an iterative execution of a set of partially ordered subtasks. Each ATR application in this class is a production job that is executed repeatedly. Thus, it is worthwhile to invest off-line time in preparing an effective mapping of the application onto the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 177 of 411

hardware platform used to execute it. The ATA (automatic target acquisition) system described in [5] and the tracking system described in [18] are examples of such iterative ATR applications.

Each application task will be an instantiation of a *DDG* (data dependency graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies among subtasks. The expected number of subtasks is 10 to 50. The DDG will be structured as a directed acyclic graph (*DAG*). The IOS is being designed for applications that will iteratively execute such a DDG. Note that while the subtasks' dependencies are represented as a DAG, subtasks themselves may contain loops.

For the initial iteration through the set of subtasks, the IOS will use information about the processing environment in its selection of algorithms for the subtasks, and their associated implementations. As part of this, the IOS will decide how to assign resources (e.g., processors) to the subtasks.

After each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image stream being processed. It is expected that the values of these parameters will change slowly. After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel. The HC Kernel will use the most recent values of such dynamic parameters to estimate if it is worthwhile to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration. If it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks. If not, the same assignment will continue to be used.

3. Heterogeneous parallel hardware platform

The type of target hardware platforms considered for this study are driven by the expected needs of the classes of ATR applications that are of interest to the U.S. Army Research Laboratory. Thus, for the intended application environment of the IOS, it is assumed that there will be up to four different types of processors, and up to a total of 64 processors (of all types combined). For example, two types may be SHARC DSP processors and PowerPC RISC processors [8]. These processors will comprise the heterogeneous parallel architecture onto which application tasks will be mapped. A system of this size should provide the real-time computing power needed for the intended application domain. The IOS approach is appropriate for larger HC platforms as well.

A small-scale example of the type of hardware platform being considered is the one described in [6]. This system was developed by the Army Research Laboratory for a real-time ATR relational template matching algorithm. Another example, although outdated, is [1], which describes a heterogeneous hardware platform designed to perform ATR research and prototyping.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 178 of 411

All the processors of all types will have communication paths to one another. Communications among processors of the same type are assumed to be *symmetric* in the sense that the conflict-free time for any pair of processors to communicate is the same. For example, a Mercury daughter board can be populated with six SHARC processors that physically share a DRAM. To connect processors of different types, a VME bus can be used to provide communications among different collections of processors. In addition, the hardware platform will include (1) a workstation, for off-line IOS operations to develop an application implementation and for use as the *Application User* interface, and (2) a *Host Processor*, which will monitor the application implementation during its execution and implement the on-line HC Kernel.

For simplicity, it is assumed that if an implementation of a given subtask uses multiple processors, all processors will be of the same type. Given this and the symmetry property of the inter-processor communications among processors of the same type, the expected execution time of a particular multiprocessor implementation of a subtask is independent of which fixed-size subset of the processors of a given type are assigned to execute the subtask.

The IOS design should be capable of working with any hardware platform of the type described above. A given hardware platform may be used for many different ATR applications. It is assumed that when a given ATR application is executing on a platform, that platform is dedicated to that application.

4. Overview of IOS off-line components

Figure 2 shows the off-line components of the IOS. The Knowledge Base contains a collection of DDGs. The Algorithm Database contains information about algorithms that can be used to perform the subtasks in the DDGs. The DDGs and algorithm information is supplied by the application domain expert, the *Application Developer*. The Application Developer, who is responsible for developing the application software, will typically be a different person (or people) from the Application User(s), who may know nothing about software development and may just use the ATR system as a prepackaged tool. An analogy to this in the personal computing field is the programmer who develops a software package for graphics versus the package user who draws figures with the tool without any knowledge of the details of the actual code in the software package.

To design a particular application, the Application Developer first selects a DDG. (The Application Developer can also specify a particular set of DDGs that can execute simultaneously and be treated as a single DDG [3].) Each DDG has associated with it a list of *Application Characteristic* and *Input Data Characteristic* names, whose values must be filled in when the DDG is instantiated for a given application and associated environment. (The SmartNet project uses a similar set of characteristics called "Compute Characteristics," whose values when an application is invoked are called the "Compute Characteristics Operating Point" [12].) As discussed in Section 6, the Algorithm Database will contain information about the expected execution time of each subtask algorithm implementation, and this time will be a function of the values of these characteristics (and other factors).

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 179 of 411

Each of these sets of Application and Input Data Characteristics can be divided into static and dynamic parameters. *Static parameters* are those Input Data Characteristics, such as image size, and Application Characteristics, such as type of object of interest (e.g., tank), that will not change during the execution of the application task. *Dynamic parameters*, in contrast, are those Input Data Characteristics, such as amount of clutter, and Application Characteristics, such as number of located objects to be identified, that will change during run time and can be computed by the application as it executes. When an Application Developer instantiates a DDG to implement an application in a given environment, the values for the static characteristics are known and provided by the Application Developer. For the dynamic characteristics, the Application Developer is expected to define a small set of dynamic parameters that will have the most impact on the execution time of the subtasks in the application. It is the job of the *Scenario Generator* component of the IOS to use information provided by the Application Developer to derive representative values for these dynamic parameters.

There are several ways in which the Scenario Generator can develop the representative dynamic parameter values. The sample ways to be described differs in the amount of involvement by the Application Developer, who is the domain expert who knows the most about the application and its associated subtasks. Also, recall from Section 2 that the values of these dynamic parameters are expected to change slowly, i.e., it is expected that most dynamic parameters will not continuously have dramatic changes in value from each image frame to the next.

Assume D dynamic parameters are selected for a given application. Let the number of representative choices for the i-th dynamic parameter be C_i, $0 \leq i < D$. Each set of D representative values for these D dynamic parameters, one per parameter, is called a *scenario*. The number of different scenarios that can be generated is $S = C_0 \times C_1 \times \cdots \times C_{D-1}$. The importance of picking an effective set of dynamic parameters and representative choices for parameter values is discussed later.

The best way to determine the representative choices for each dynamic parameter is to have the Application Developer specify them. The Scenario Generator can be set up to work interactively with the Application Developer to accomplish this (through the *IOS-App Builder* software component of the IOS).

Consider the following hypothetical example. The task is the tracking of vehicles on the ground, and the Application Developer has chosen four dynamic parameters: (1) number of objects of interest being tracked in the last image frame, (2) average size of the objects currently being tracked (in terms of percent of the total number of pixels in the image), (3) whether the lighting conditions in the last image frame required requantization for better contrast for display, and (4) whether there was cloud cover detected in the last image frame that required special processing. The representative choices specified for the dynamic parameters are: (1) 0, 1, 2, 4, 8, 12, (2) 5%, 10%, 15%, 20%, 40%, (3) yes or no, and (4) yes or no. Thus, $C_0 = 6$, $C_1 = 5$, $C_2 = 2$, $C_3 = 2$, and $S = 120$.

This example demonstrates another aspect of the scenario generation process: pruning. In some cases, there may be parts of the D-dimensional dynamic parameter space that have a low probability of occurring in reality. For this example, having the number of objects being tracked be eight or twelve, and having the average size of an object be 40%,

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 180 of 411

is very unlikely. The Scenario Generator will work interactively with the Application Developer (through the IOS-App Builder) to prune out such combinations of dynamic parameter values. Even if a certain improbable combination that is pruned does appear during the execution of the application, the HC Kernel will select the combination that was included that is closest to the actual run-time values of the dynamic parameters.

An alternative to having the Application Developer specify the exact representative choices for each dynamic parameter is to have the Application Developer just specify a range of values (e.g., minimum and maximum) and the associated C_i for each dynamic parameter. The Scenario Generator could then calculate a set of C_i uniformly distributed choices over the specified range. Of course, any combination of these two approaches could also be used.

Thus, the Scenario Generator constructs S scenarios, where each scenario corresponds to a particular set of representative values for the D dynamic parameters (one representative value for each dynamic parameter). The Scenario Generator passes these S scenarios to the off-line component of the ATR Kernel, as shown in Figure 2.

The job of the off-line component of the ATR Kernel is to select one or more algorithms from the Algorithm Database for each subtask of the DDG previously instantiated for the given application by the Application Developer. This is done separately for each of the S scenarios associated with the given application. The algorithm selection is based on the ability of a given algorithm to perform the image processing required by a given subtask; i.e., each subtask is assigned one or more algorithms whose *image processing capability* for that subtask and its associated Application and Input Data Characteristics is above some threshold. Thus, the selection process can use information about the application's static parameter values and that scenario's dynamic parameter values. The execution time of an implementation of the algorithm is not considered by the ATR Kernel (the HC Kernel handles this).

Performing this assignment of one or more algorithms to each subtask transforms a DDG into an *ODDG* (over-instantiated DDG), and the off-line component of the ATR Kernel is called the *ATR Kernel ODDG Generator*. Therefore, for a given application, the ATR Kernel ODDG Generator creates a distinct ODDG for each scenario.

In summary, for a given application, the Application Developer specifies a DDG, an associated static environment, and D dynamic parameters (with representative values). The Scenario Generator creates S scenarios based on the representative values of the dynamic parameters. The ATR Kernel ODDG Generator constructs S ODDGs for the given application, one for each scenario.

For each ODDG, one *MDDG* (mapped DDG) is created by the *HC Kernel MDDG Generator*. The MDDG specifies how the corresponding ODDG will be implemented and mapped onto the HC platform, as discussed further in Section 6. Therefore, S MDDGs are generated by the HC Kernel MDDG Generator for a given application and its associated static environment. These S MDDGs are sent by the HC Kernel MDDG Generator to the *MDDG Table Builder*, which constructs the *MDDG Table* for that application and its associated static environment. The MDDG Table will be indexed as a D-dimensional array, where the i-th dimension is of size C_i, i.e., an MDDG Table entry will be accessed by a list of D indices (corresponding to a scenario), where the value of the i-th index

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 181 of 411

corresponds to one of the C_i allowable representative values for the i-th dynamic parameter. MDDG Tables are stored in the Knowledge Base, as are the associated *Application Menus* constructed by the *Application Menu Builder*.

A very important question that arises is what set of dynamic parameters and representative values for these parameters should be used in the scenario generation process. The number of scenarios, S, should also be considered. The larger S is, the closer a given scenario in the MDDG Table may match a given set of actual dynamic parameter values calculated during execution of the application. It is expected that the closer this match is, the better the mapping specified by the corresponding MDDG Table entry will be. However, the larger S is, the larger the MDDG Table will be, because it will hold S MDDGs. Also, increasing S increases the off-line execution time for the ATR Kernel ODDG Generator and HC Kernel MDDG Generator; however, for the intended production ATR application domain, the off-line preparation time is much less significant than the resulting on-line execution time of the application task. The Application Developer will need to trade off these factors when selecting dynamic parameters and representative choices for these parameters, based on a knowledge of the problem domain and the sensitivity of the expected execution times of the subtasks to the values of these parameters. Thus, these decisions will depend on the given application, static parameter values, and ranges for dynamic parameter values for a given environment. In general, this may require the Application Developer to perform some experiments with sample data sets to determine the best dynamic parameters and representative values to use in order to achieve the desired system performance (i.e., use more off-line time to examine multiple sets of scenarios to determine which one will give acceptable on-line application performance using an acceptable size MDDG Table). The IOS can be designed to readily support such testing.

5. Overview of IOS on-line components

Figure 3 shows the on-line components of the IOS. Once in the field, the Application User interacts with the *ATR Kernel User Interface*, including the Application Menus. The Application Menus will be used on-line by the Application User to select a particular initial MDDG to use to invoke an application (with an associated set of Application User specified values for static parameters and initial choices of allowable representative dynamic parameters). That MDDG is passed to the *HC Kernel Monitor* to use as the initial mapping. Then, as the application is executing, the HC Kernel Monitor monitors the run-time values of the dynamic parameters at the end of each iteration through the underlying DDG to make a heuristically-based decision whether to continue with the current mapping, or to select and instantiate a new mapping (for the next iteration) from among the entries of the relevant MDDG Table (which were determined off-line). Thus, the off-line processing provides a set of predetermined mappings that the on-line processing can index in real time.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 182 of 411

6. HC Kernel MDDG Generator

The HC Kernel MDDG Generator is the component of the off-line IOS (Figure 2) that is responsible for mapping each ODDG onto the heterogeneous parallel hardware platform creating a corresponding MDDG. Three items of information about the hardware platform will be needed by the HC Kernel MDDG Generator: (P1) algorithm implementation execution time, (P2) number of each type of processor available, and (P3) inter-processor communication time. The HC Kernel MDDG Generator uses this information when it applies the heuristic for determining an effective assignment of subtask implementations to processors.

The Algorithm Database will include one or more implementations of each algorithm (e.g., one for each processor type). The Algorithm Database must also contain the expected execution time of each algorithm implementation, typically specified as a function of type and number of processors assigned, interprocessor communication time, and certain static and dynamic Input Data and Application Characteristics. This is an expected time, rather than a definite time, because it may vary depending on the actual values of the input data being processed. This expected execution time information must be provided by the Application Developer, who also supplies the code for each implementation of a given algorithm. It is expected that algorithm implementations will be written using the number of processors (of a given type) as an input parameter whenever possible. The assumption of the availability of expected implementation execution time for each type of processor (or set of processors of the same type) is typically made for the current state of the art in HC systems (e.g., [11], [13], [20], [24]). The Application Developer can determine this information empirically. The HC Kernel MDDG Generator needs to use this information (in conjunction with the other information below) to determine the expected total application task execution time.

The second item needed is the total number and type of the processors in the platform. The HC Kernel MDDG Generator needs this to know how many processors of each type it has available to assign. This is specified by the *Platform Architect*, who is responsible for the hardware design of the system. Typically, this person (or people) is distinct from the Application Developer(s) and Application User(s), although it is assumed that the Platform Architect(s) will consult with the Application Developer(s). Referring to the earlier analogy with personal computing, the people responsible for designing the system hardware configuration will typically not be the same people who develop the graphics package.

The last item needed is a *communication matrix* indicating the time it takes for each processor in the platform to communicate with every other processor in the platform. This is also specified by the Platform Architect. Entry (i, j) in this matrix is the information needed to calculate the conflict-free time for processor i to send data to processor j. The communication time for a given pair of processors typically will have two components: a fixed latency time for the first byte to arrive, and a variable time that depends on the length of the message being transmitted (that is based on the bandwidth of the communication path). This type of matrix is used by other researchers in HC (e.g., [13], [20], [24]). This matrix is needed for the HC Kernel MDDG Generator to determine the expected inter-

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 183 of 411

subtask communication times for possible mappings of subtasks onto the platform. Recall from Section 3 that it is assumed that the target hardware platform is dedicated to this application, so the communication channels can be scheduled to prevent conflicting requests.

Consider the structure of an MDDG. Each MDDG is isomorphic to an ODDG (i.e., for each node in the ODDG, there is a unique corresponding node in the MDDG). Each MDDG node stores node–specific information that includes: (S1) which implementation (stored in the Algorithm Database) will be used for one of the algorithms in the corresponding ODDG node; (S2) pointers to the needed object code for that implementation; (S3) any additional information needed for loading that implementation onto the heterogeneous parallel system; (S4) any needed inter-subtask (i.e., inter-node) communications between that given MDDG node and any other node in the MDDG (which will also enforce data-dependency constraints among the subtasks); and (S5) the specific set of processors that will be used to execute that MDDG node. In addition to this node specific information, global MDDG information is stored with each MDDG, including: (G1) the schedule for the execution order of the MDDG nodes and inter-subtask data transfers; and (G2) the expected execution time for one iteration through the MDDG for the scenario (values of dynamic parameters) that was specified along with (and is the basis of) the corresponding ODDG. The execution time information in (G2) above is used by the HC Kernel Monitor when deciding whether to change the mapping (i.e., reconfigure), as discussed in the next section. As discussed earlier, the HC Kernel MDDG Generator passes this MDDG to the MDDG Table Builder, to be stored as part of the MDDG Table for the given application and static environment characteristics.

The Application Developer can also specify an estimate of the average overhead time to reconfigure the mapping of the given application (and associated static environment) on the hardware platform. This estimate will represent the time needed to remap the application during execution as a result of changes to the values of the dynamic parameters. The estimated average reconfiguration overhead time will be sent from the Application Developer to the HC Kernel MDDG Generator through the IOS-App Builder software. If the Application Developer does not know how to estimate this value, the Application Developer can provide the IOS-App Builder with a set of scenarios that are expected to occur frequently based on experience with the application, and the IOS-App Builder can use these to actually perform remappings among these scenarios to calculate an overhead estimate. Alternatively, the IOS-App Builder can generate a relatively small random subset of scenarios to use to calculate the estimate. Rather than derive a single estimated average overhead value, the reconfiguration time could be calculated for each of the S^2 possible old and new configuration pairs, and stored in the Knowledge Base; however, this would require an excessive amount of space to store and an excessive amount of off-line time to calculate, and, thus, it is not advisable.

The HC Kernel MDDG Generator will pass the average reconfiguration overhead time estimate to the MDDG Table Builder to be stored as part of the MDDG Table for this application. This overhead time will be used by the HC Kernel Monitor when deciding whether or not to perform a reconfiguration.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 184 of 411

Thus, the HC Kernel MDDG Generator gets from the ATR Kernel ODDG Generator an ODDG and an associated scenario, it creates an MDDG containing the information specified above, and then passes this MDDG to the MDDG Table Builder that constructs a complete table and stores it in the Knowledge Base. The rest of this section will examine how the HC Kernel MDDG Generator can derive the information that comprises the MDDG.

In the HC field, the MDDG node specific items (S1), (S4), and (S5) defined earlier in this section are part of the process of matching subtasks in a task graph to processors in the heterogeneous parallel system. The MDDG node specific items (S2) and (S3) above are adaptations of standard operating system functions that will need to be implemented, but will not be discussed further here. The global MDDG item (G1) is referred to as the scheduling component of a mapping in an HC environment.

For general HC, deriving an optimal matching and scheduling is intractable (i.e., it is known to be an NP-complete problem that requires exponential execution time to perform an exhaustive search of the space of possible solutions [10]). This is true even when all execution and communication times can be determined statically (i.e., they are not input-data dependent). Consider the intended application domain and hardware platforms. As an example, assume there are four types of processors, with 16 processors of each type, and assume the application consists of 50 subtasks. Then, an exhaustive search of all possible assignments of subtasks to machines will take time proportional to $(4 \times 16)^{50}$. Thus, a heuristic is used, as is common in the heterogeneous computing field [22], [23].

The structure of the HC Kernel MDDG Generator is such that any good heuristic could be employed. The heuristic that is used could be one that can be executed during run time, such as a levelized-min-time heuristic (e.g., [16]), or an off-line heuristic, such as a genetic algorithm (e.g., [20], [24], [28]). In general, an off-line heuristic can find a better mapping than an on-line run-time heuristic because its execution time can be orders of magnitude longer than that of the run-time algorithm. An example of this difference in quality of matchings is provided in [28]. However, because of the longer execution times of genetic-algorithm-based heuristics, it would not be appropriate to execute a genetic algorithm while a real-time application is running in order to decide how to reconfigure resources based on the actual values of the dynamic parameters at the end of a given iteration through the underlying DDG.

For this application domain, it is possible to use off-line precomputed mappings to reconfigure resources in real time. In particular, the IOS will: (I1) allow the HC Kernel MDDG Generator to use genetic algorithms (or other off-line heuristics) to determine a matching and scheduling (i.e., MDDG) off-line for each scenario associated with a production ATR application task; (I2) allow the Application User to select an initial MDDG when the application's execution is initiated (through the ATR Kernel User Interface); and (I3) allow the HC Kernel Monitor to select a new MDDG during execution if desired based on the actual values of the dynamic parameters at the end of an iteration through the corresponding DDG. Item (I1) above is the subject of this section, item (I2) is part of the ATR Kernel User Interface (not the HC Kernel), and item (I3) will be covered in the next section.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 185 of 411

While the HC Kernel MDDG Generator can incorporate any appropriate off-line mapping heuristic, to describe the design ideas involved in the HC Kernel it will be assumed that a genetic algorithm is used. *Genetic algorithms* are a promising heuristic approach to optimization problems that are intractable [7], [14], [15]. There are a great variety of approaches to genetic algorithms; many are surveyed in [19], [25].

In this environment, the genetic algorithm is a guided heuristic search through the space of possible matchings and schedulings (called *solutions*). The genetic algorithm in [28] was found to be very successful, and used an HC model that is quite compatible with the situation here. However, in [28] each subtask was assigned a single machine, so, for the type of platforms described in Section 3, the "chromosome" representing the matching will have to be adapted to allow for multiple processors of the same type to be assigned to a subtask. A technique for doing this has been developed [2], but its discussion is outside the scope of this paper.

The approach in [28] differs from other genetic algorithm approaches to matching and scheduling for heterogeneous systems in the literature ([20], [24], [27]) in many ways. The most significant difference from [20] and [24] is that the [28] model of a heterogeneous system is more realistic (e.g., [24] assumes an unlimited number of machines). The main difference between [28] and [27] is that in [28] it is assumed that there is a given target hardware platform, whereas [27] selects processors to be included in the platform and uses processor cost as a co-metric. Thus, for the HC Kernel for the application and platform environments discussed in Sections 2 and 3, it is most appropriate to build on the [28] genetic approach.

For the intended application domain here, the genetic algorithm needs the following information to create a matching and scheduling, i.e., to transform an ODDG into an MDDG: (M1) the structure of the underlying DDG; (M2) the expected execution time of each subtask on a set of processors (of the same type) assigned, as a function of the type of the processors and the number of processors; (M3) the inter-subtask data transfers needed, in terms of formats and expected sizes of the data items to be transferred; (M4) the expected time to send data from one processor to another as a function of the size of the data item to be transferred; and (M5) the number of each type of processor that is in the hardware platform. The genetic algorithm selects a subset of possible solutions and then evaluates them using the information in items (M1) to (M5) above. The genetic algorithm uses the results of the evaluations of these possible solutions to generate a new set of possible solutions. This process iterates until one of the stopping criteria is met (e.g., no improvement in solution quality after a given fixed number of iterations within the genetic algorithm). When the genetic algorithm stops iterating, the best solution found is used as the mapping for the MDDG.

How the HC Kernel MDDG Generator gets each of the above information items is now considered. Much of the relevant information flow is depicted in Figure 2.

The HC Kernel MDDG Generator gets the item (M1) from the ODDG that is passed to it from the ATR Kernel ODDG Generator. The genetic algorithm uses a topological sort (i.e., a valid total ordering) of the subtasks in the ODDG to establish the order in which it evaluates the nodes of the ODDG.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 186 of 411

Item (M2) is provided by the Application Developer and stored in the Algorithm Database. Information about static and dynamic parameters may be needed in some cases (e.g., the size of subtask input data blocks). The HC Kernel MDDG Generator will receive the static information from the IOS-App Builder and the dynamic (scenario) information from the ATR ODDG Generator (see Figure 2). The expected execution times for subtasks calculated from this information are used by the genetic algorithm in its evaluation of possible solutions.

Recall that each subtask of the underlying DDG corresponds to a node of the ODDG that may contain more than one possible algorithm to perform the subtask. For each of these algorithms, there is at least one implementation that can execute on the hardware platform, and possibly more than one. When the genetic algorithm evaluates a given possible mapping solution (i.e., assignment of resources), it selects the implementation for the subtask that has the smallest expected execution time for the resources assigned to that subtask. If no implementation is available for that assignment, that mapping is considered invalid by the genetic algorithm and is discarded from the set of possible mappings.

There may be cases when there are inter-subtask implementation interactions that must be considered when selecting the implementation for a given subtask. One example of how this can occur is when some subtask *A* sends a data block, e.g., an image, to a subtask *B*, and the image is stored using different formats for the two subtasks' implementations selected (e.g., assigning rectangular subimages to processors in a multiprocessor implementation for subtask *A* and row striping of subimages in a multiprocessor implementation for subtask *B*). When this occurs, overall implementations based on each data format are considered, and the one with the smallest execution time is used as a basis for selecting the implementations for subtasks *A* and *B* for this possible solution mapping. For the intended application domain, this should not cause a significant time penalty relative to the total execution time for the genetic algorithm to generate a mapping for the MDDG being constructed.

An alternate approach for handling mismatched data block storage organizations is to allow each subtask implementation to use its own choice of organization and convert between organizations during run time. For the intended real-time ATR applications, this option will not be considered; however, if it becomes desirable to consider this option in the future, the genetic algorithm framework of [28] will allow it to be included.

Item (M3) is provided by the Application Developer and stored in the Algorithm Database as part of the input/output parameter descriptions (see [2], [3]). This inter-subtask data transfer information is used by the genetic algorithm (in conjunction with the information in item (M4)) in its evaluation of possible solutions.

Item (M4) is provided by the Platform Architect and stored in the Knowledge Base as part of the hardware description. Together with the information from item (M3), the genetic algorithm can evaluate the expected inter-subtask communication times for a given possible mapping solution.

Item (M5) is provided by the Platform Architect and stored in the Knowledge Base as part of the hardware description. It is used by the genetic algorithm to know the upper bound on the number of each type of processor that can be assigned. It should be noted that in some cases the optimal mapping (in terms of total execution time for an iteration

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 187 of 411

through the ODDG) may not use all available processors. That is, in some cases, the overhead involved in using all of the processors may result in a longer total execution time for an iteration than if only a subset of the available processors are used, as discussed in [21].

Thus, using all of this information, the genetic algorithm can create a matching and scheduling to be included in the MDDG for the given ODDG. As part of the evaluation for the potential solutions, the genetic algorithm computes the expected total execution time per iteration for each solution. For the mapping solution chosen, this expected total time is stored with the MDDG, as mentioned earlier. The completed MDDG is sent to the MDDG Table Builder.

The HC Kernel can also be used to provide some fault tolerance for the overall system. To accomplish this, the Platform Architect first specifies the fault-free system as the target platform. The application is processed using the IOS methodology presented here, forming an MDDG Table for that application and target platform. After this is done, the Platform Architect can specify as the target platform a faulty variation of the fault-free system, e.g., with one fewer functioning type "Y" processor. The application is processed again, this time using the faulty variation of the system as the target platform, forming a second MDDG Table, which corresponds to the same application but using the faulty variation of the system as the target platform. By doing this, the HC Kernel MDDG Generator will construct an MDDG Table of mappings that can be accessed and used in real time should a hardware fault of the type planned for occur. For the example above, if a type "Y" processor becomes faulty, the HC Kernel will switch from using the fault-free case MDDG Table and use the MDDG Table corresponding to this single-fault case instead.

This can be done for different faulty variations of the hardware system. Each MDDG Table will correspond to a given platform variation for that application with its associated static environment. Obviously, each variation considered increases the off-line computation and the storage that will be needed. The trade-off of this extra off-line time to prepare these MDDG Tables and the storage to keep them against the importance of handling faults for a given operational situation must be considered.

Given the symmetric assumption of Section 3, if an MDDG Table is prepared for the case of a faulty type "Y" processor, that MDDG Table can handle any one of the type "Y" processors becoming faulty. Thus, if one faulty variation on the hardware system is generated for each type of processor (e.g., four, as discussed in Section 3), then the case of any one processor of any type becoming faulty can be handled.

7. HC Kernel Monitor

The HC Kernel Monitor is the on-line component of the IOS (Figure 3) responsible for (1) establishing the initial mapping of the given application onto the hardware platform, and (2) monitoring the execution of the application and, at the end of each iteration through the corresponding DDG, deciding if and how the mapping of the application onto the hardware platform should be changed based on information about the actual values of the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 188 of 411

dynamic parameters. To establish the initial mapping, the HC Kernel Monitor uses the MDDG index (scenario) and associated MDDG Table identifier passed to it from the ATR Kernel User Interface. With this index and identifier, the HC Kernel Monitor can access from the appropriate MDDG Table in the Knowledge Base the MDDG entry selected by the Application User (see Figure 3). The HC Kernel Monitor then passes the relevant information to the Basic Kernel (i.e., loading, configuring, and scheduling information). The Basic Kernel accesses the implementation code directly from the Algorithm Database using pointers provided with the loading information. The Basic Kernel can then begin execution of the application task.

During execution of the application, the HC Kernel Monitor receives updated actual values of the dynamic parameters at the end of each iteration through the corresponding DDG. Specifically, after all subtasks' implementations have completed execution for a given iteration of the DDG corresponding to the application, and before the next iteration begins, the latest values of these dynamic parameters will be sent to the HC Kernel Monitor from the Basic Kernel. The HC Kernel Monitor will use the most recent values of these dynamic parameters to estimate if changing the matching and scheduling will reduce the expected execution time of the next iteration through the corresponding DDG. Thus, this decision is made in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration.

If it is desirable to change the mapping, then it is the responsibility of the HC Kernel Monitor to select a new matching and scheduling (i.e., a new MDDG entry from the given MDDG Table) to use for the next execution iteration through the corresponding DDG. If not, the same mapping will continue to be used.

To determine if the current mapping should be changed, the HC Kernel Monitor performs the following sequence of steps.

(1) For each dynamic parameter, the representative value included in the allowable choices for that parameter (as specified by the Scenario Generator) is found that is closest in absolute difference to the current actual value for that parameter. This is done for all D dynamic parameters for this application. The resulting vector of D representative values is used as the approximation of the set of current actual dynamic parameters in terms of a precomputed scenario (MDDG Table index). Call this new MDDG Table index A. If MDDG Table index A is the same MDDG Table index as the one used for the current iteration, the next iteration of the application proceeds with the same mapping as the current iteration. If MDDG Table index A is not the same as the one used for the current iteration, then steps (2) through (5) are performed.
(2) The HC Kernel Monitor then accesses the MDDG Table for this application, using A as the index. The expected execution time for an iteration through the MDDG Table entry corresponding to MDDG Table index A is used as an estimate of what the expected execution time will be for the actual dynamic parameters for the next iteration of the application if the remapping occurs. Call this time $T1$.
(3) The actual execution time of the last (current) iteration through the application's corresponding DDG is used as an estimate for the time to execute the next iteration

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 189 of 411

with the current actual dynamic parameters with the current mapping (this is only an estimate because some or all of these parameter values may have been different at the beginning of the current iteration and changed sometime during the execution of this iteration). Call the actual execution time for the current iteration *T2*.

(4) When making the decision to reconfigure the resources, in addition to comparing *T1* and *T2*, the HC Kernel Monitor must also consider the time required to perform the reconfiguration (e.g., any code and data movements that the reconfiguration will require). The average reconfiguration overhead time estimate stored with the MDDG Table is used. Call this overhead time *TO*. (The HC Kernel Monitor could start with the estimate provided with the MDDG Table and then, for each time a reconfiguration is performed for this application, measure the actual overhead time and use this experiential information to modify the most recent estimate in some weighted way.)

(5) If $(T1 + TO) < T2$, then the HC Kernel Monitor instructs the Basic Kernel to change the mapping to the one corresponding to the MDDG Table entry for MDDG Table index *A*.

Thus, the HC Kernel Monitor can use the actual values of dynamic parameters at the end of each execution iteration of the application to make remapping decisions and select new mappings derived by a time-consuming off-line heuristic. As can be observed, the execution of the computationally simple steps (1) to (5) above can be done in real time, causing relatively negligible overhead compared to the expected execution time of an iteration through an ATR DDG.

The decisions made by the HC Kernel Monitor are based on heuristics and approximations. Thus, pathological cases could cause a bad decision to be made. However, pathological worst cases will also exist for fixed mappings (based on average performance) and for on-line schedulers. In general, the ideas underlying the HC Kernel Monitor will lead to reduced overall application execution time for the environment under consideration. Obviously, the HC Kernel methodology should be used only in those situations where it proves effective compared to other approaches, which can be determined only by analysis and experimentation for the exact application and target hardware platform under consideration.

As an example of a possible pathological case that could occur, if $T1 = T2 - i$ and $TO = i + 1$, then $(T1 + TO) > T2$, and reconfiguration would not be done. If the values of the dynamic parameters do not change for the next 20 iterations, the execution time for those iterations will total approximately $20 \times T2$. If reconfiguration had been done, the execution time for those iterations would total approximately $20 \times T1 = 20 \times T2 - 20 \times i$. Because the values of dynamic parameters depend on the input data, the number of iterations that will occur with no changes to the dynamic parameter values can never be predicted with certainty. If situations such as the above do appear to occur frequently for a given application, it would be possible to instrument the HC Kernel Monitor to collect the relevant statistics and then develop and add special rules. For this example, a new rule may be that if after so many iterations with no change to the dynamic parameter values, if the relationship between *TO* and $(T2 - T1)$ is smaller than some threshold, perform the remapping. However, there is no guarantee that the dynamic parameter values will not

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 190 of 411

change during the next iteration, so the decision to include such rules must be made carefully.

Thus, by studying properties of the application domain, the IOS builder may decide to fine tune the HC Kernel Monitor in different ways. The HC Kernel design has the flexibility to allow such tuning.

8. Summary

This study focused on a design for an HC Kernel for iterative ATR tasks and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, an HC Kernel was presented for making real-time on-line input-data dependent remappings of the application subtasks to the processors in the heterogeneous parallel hardware platform using previously stored off-line statically determined mappings. In particular, it was shown how the HC Kernel can be used to create the MDDG Table off-line and then use it to make real-time on-line decisions and selections of mappings. In addition to the HC Kernel, the relevant parts of other components of the IOS were briefly discussed. The overall strategy of the IOS and the interactions of the IOS components are summarized in Figures 1 to 3. The HC Kernel ideas introduced here can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the applications and platforms considered here.

The goal of this paper has been to introduce the concepts of the HC Kernel, explain when it can be applied, and to give a framework for developing it. Future research, by our team and other groups, includes building HC Kernel based systems for a variety of different applications and a variety of different target hardware platforms to determine where the HC Kernel methodology can be best exploited.

Acknowledgments

The authors thank Janet M. Siegel, Min Tan, Tracy Braun, Ricky Yu-Kwong Kwok, and the anonymous reviewers for their comments. A preliminary version of this paper was presented at the *6th Heterogeneous Computing Workshop*.

This research is supported by the Army Research Laboratory, under contract number DAAL01-96-C-0031. H. J. Siegel is a consultant for Architecture Technology Corporation.

References

1. S. Brandt and J. R. Budenske, "Starcon—a reconfigurable fieldable signal processing system," *SPIE Conf. on Image Understanding in the '90s: Building Systems that Work, SPIE Vol. 1406*, pp. 122–126, Oct. 1990.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 191 of 411

2. J. R. Budenske, H. J. Siegel, R. S. Ramanujan, K. J. Thurber, and M. D. Pritt, *Intelligent Operating System Final Technical Report*, Architecture Technology Corp., Minneapolis, MN, Technical Report ATC-RD-96-04, 91 pp., Oct. 1996.
3. J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, "'Modeling ATR applications for intelligent execution upon a heterogeneous computing platform," *1997 IEEE International Conf. on Systems, Man, and Cybernetics (SMC '97)*, pp. 649–656, Oct. 1997.
4. C. H. Chu, E. J. Delp, L. H. Jamieson, H. J. Siegel, F. J. Weil, and A. B. Whinston, "A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," *Journal of Parallel and Distributed Computing*, Vol. 6, No. 3, pp. 598–622, June 1989.
5. P. David, S. Balakirsky, and D. Hillis, "A real-time automatic target acquisition system," *Conf. on Unmanned Vehicle Systems*, pp. 183–198, July 1990.
6. P. David, P. Emmerman, and S. Ho, "A scalable architecture system for automatic target recognition," *13th AIAA/IEEE Digital Avionics Systems Conf.*, pp. 414–420, Oct. 1994.
7. L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, NY, 1991.
8. T. H. Einstein, "Mercury Computer Systems' modular heterogeneous RACE multicomputer," *6th Heterogeneous Computing Workshop (HCW '97)*, pp. 60–71, Apr. 1997.
9. M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.
10. D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, pp. 1427–1436, Nov. 1989.
11. R. F. Freund, "The challenges of heterogeneous computing," *Parallel Systems Fair at the 8th Int'l Parallel Processing Symp.*, pp. 84–91, Apr. 1994.
12. R. F. Freund, T. Kidd, D. Hensgen, and L. Moore, "SmartNet: A scheduling framework for heterogeneous computing," *2nd Int'l Symp. on Parallel Architectures, Algorithms, and Networks (I-SPAN '96)*, pp. 514–521, June 1996.
13. A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, pp. 78–86, June 1993.
14. D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
15. J. H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, Ann Arbor, MI, 1975.
16. M. A. Iverson, F. Ozguner, and G. J. Follen, "Parallelizing existing applications in a distributed heterogeneous environment," *4th Heterogeneous Computing Workshop (HCW '95)*, pp. 93–100, Apr. 1995.
17. A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, pp. 18–27, June 1993.
18. M. B. Kulaczewski and H. J. Siegel, "Implementations of a feature-based visual tracking algorithm on two MIMD machines," *1997 International Conf. on Parallel Processing (ICPP '97)*, pp. 431–440, Aug. 1997.
19. J. L. Ribeiro Filho and P. C. Treleaven, "Genetic-algorithm programming environments," *IEEE Computer*, Vol. 27, No. 6, pp. 28–43, June 1994.
20. P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, "Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 98–117, Apr. 1996.
21. H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer-vision-related tasks onto reconfigurable parallel processing systems," *IEEE Computer*, Vol. 25, No. 2, pp. 54–63, Feb. 1992.
22. H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and Y. Alexander Li, "Heterogeneous computing," in *Parallel and Distributed Computing Handbook*, ed. A. Y. Zomaya, McGraw-Hill, New York, NY, pp. 725–761, 1996.
23. H. J. Siegel, H. G. Dietz, and J. K. Antonio, "Software support for heterogeneous computing," in *The Computer Science and Engineering Handbook*, ed. A. B. Tucker, Jr., CRC Press, Boca Raton, FL, pp. 1886–1909, 1997.
24. H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 86–97, Apr. 1996.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 192 of 411

25. M. Srinivas and L. M. Patnaik, "Genetic algorithms: A survey," *IEEE Computer*, Vol. 27, No. 6, pp. 17–26, June 1994.
26. V. S. Sunderam, "Design issues in heterogeneous network computing," *Workshop on Heterogeneous Processing*, revised edition, pp. 101–112, Mar. 1992.
27. Y. G. Tirat-Gefen and A. C. Parker, "MEGA: An approach to system-level design of application specific heterogeneous multiprocessors," *5th Heterogeneous Computing Workshop (HCW '96)*, pp. 105–117, Apr. 1996.
28. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Computing*, Special Issue on Parallel Evolutionary Computing, Vol. 47, No. 1, pp. 8–22, November 1998.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 193 of 411

US 20180074855A1

(19) **United States**

(12) **Patent Application Publication**

(10) **Pub. No.: US 2018/0074855 A1**

(43) **Pub. Date: Mar. 15, 2018**

Kambatla

(54) **UTILIZATION-AWARE RESOURCE SCHEDULING IN A DISTRIBUTED COMPUTING CLUSTER**

(71) Applicant: **Cloudera, Inc.**, Palo Alto, CA (US)

(72) Inventor: **Karthik Kambatla**, Mountain View, CA (US)

(21) Appl. No.: **15/595,713**

(22) Filed: **May 15, 2017**

Related U.S. Application Data

(60) Provisional application No. 62/394,660, filed on Sep. 14, 2016.

Publication Classification

(51) **Int. Cl.**

G06F 9/48 (2006.01)

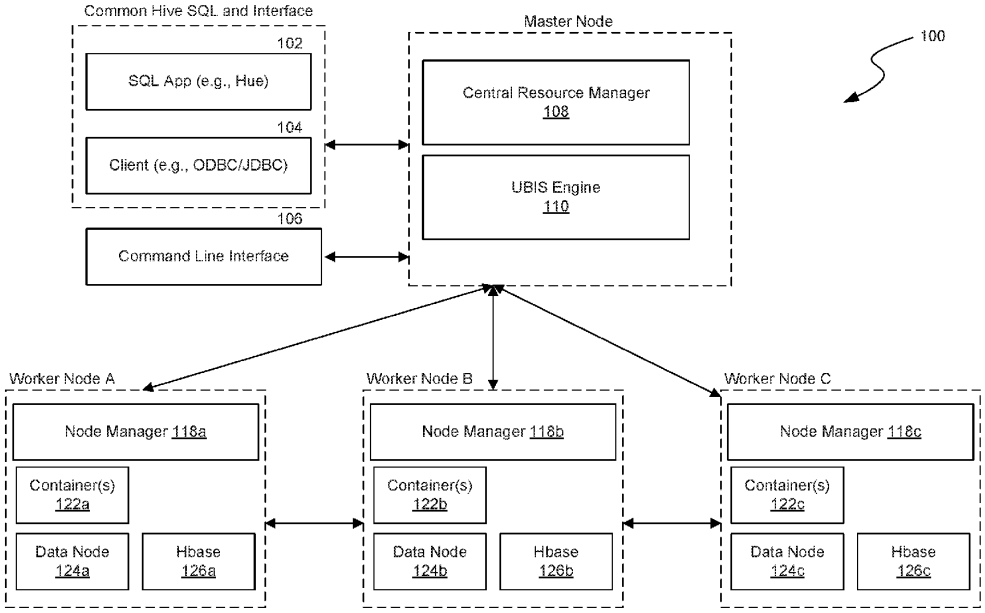
G06F 9/50 (2006.01)

(52) **U.S. Cl.**

CPC ***G06F 9/4881*** (2013.01); ***G06F 9/5083*** (2013.01); *G06F 2209/483* (2013.01); *G06F 2209/5021* (2013.01); ***G06F 9/5038*** (2013.01)

(57) **ABSTRACT**

Embodiments are disclosed for a utilization-aware approach to cluster scheduling, to address this resource fragmentation and to improve cluster utilization and job throughput. In some embodiments a resource manager at a master node considers actual usage of running tasks and schedules opportunistic work on underutilized worker nodes. The resource manager monitors resource usage on these nodes and preempts opportunistic containers in the event this over-subscription becomes untenable. In doing so, the resource manager effectively utilizes wasted resources, while minimizing adverse effects on regularly scheduled tasks.



Darell Long, Ph.D.
09/17/2025
Exhibit E
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1006

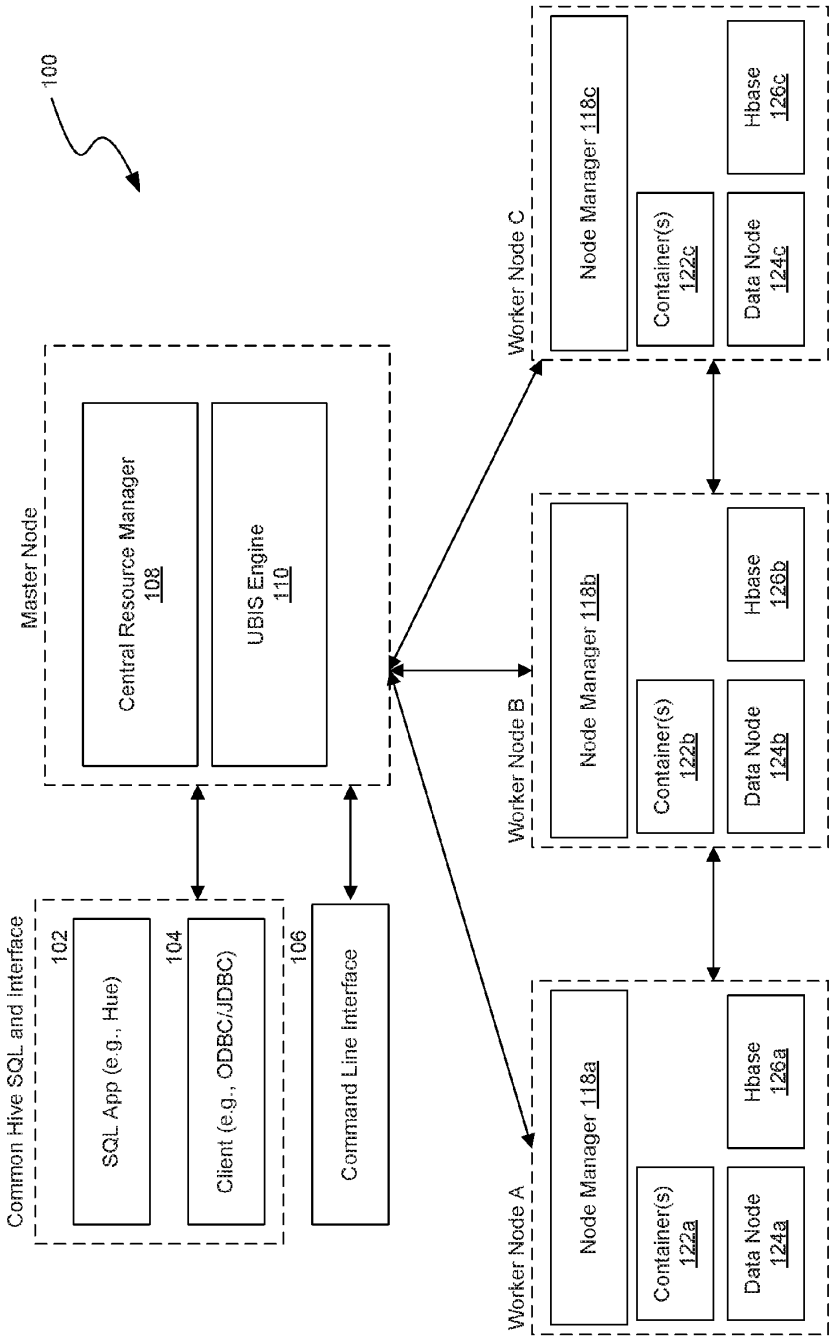


FIG. 1

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 195 of 411

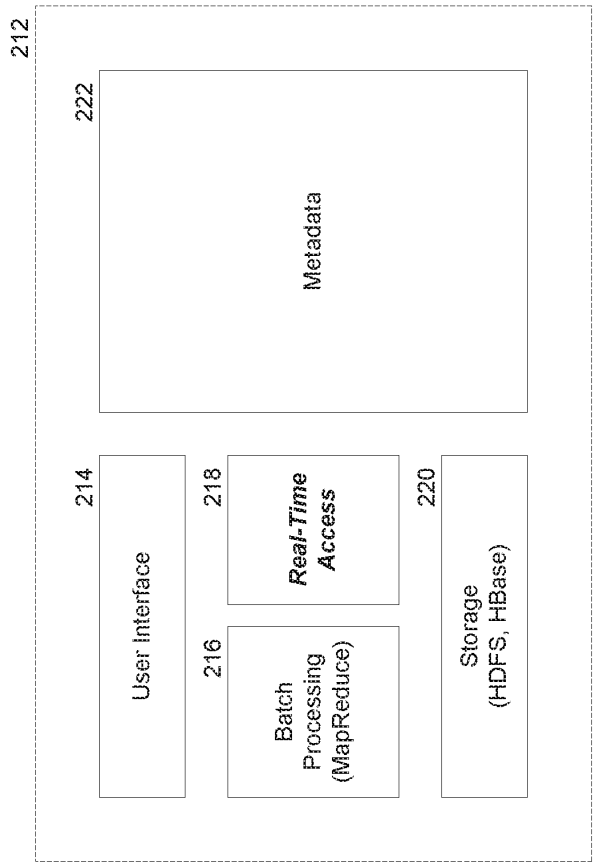


FIG. 2

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 196 of 411

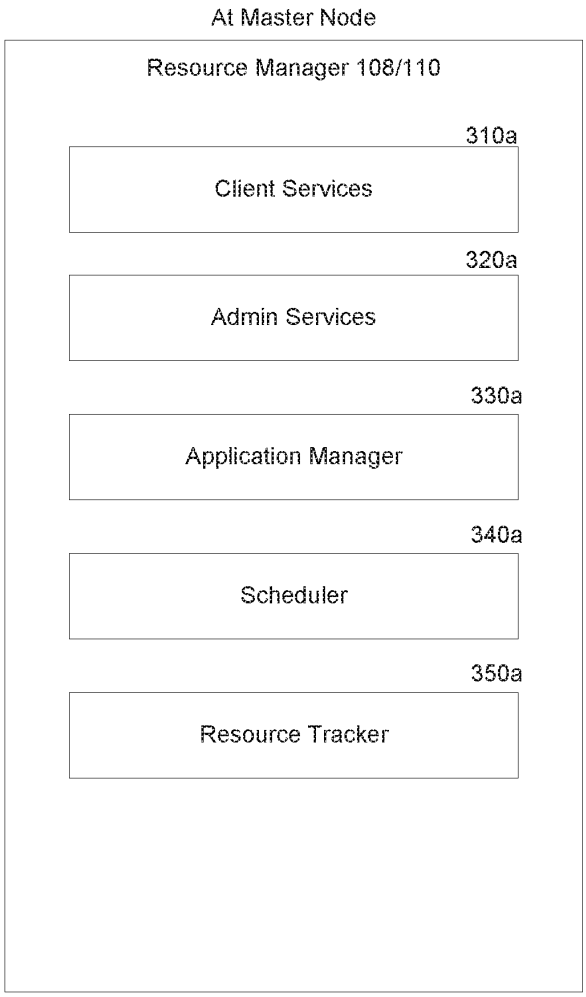


FIG. 3A

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 197 of 411

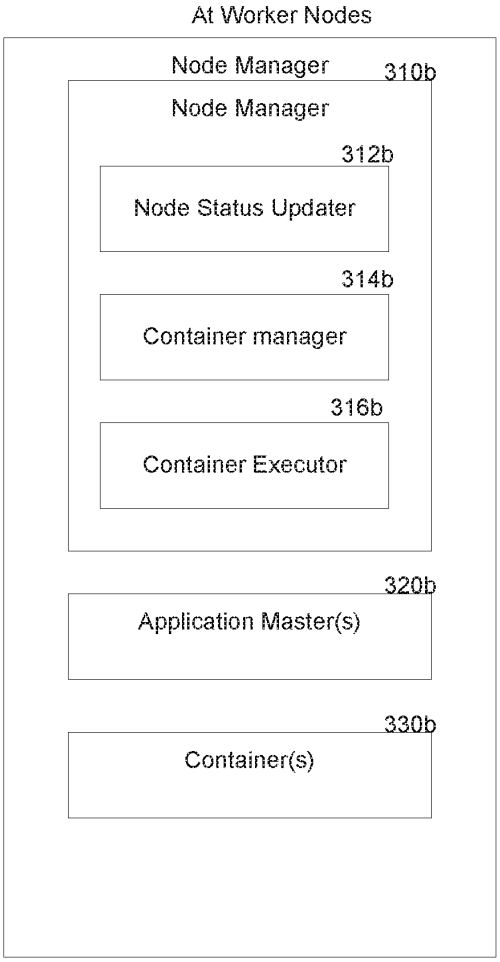


FIG. 3B

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 198 of 411

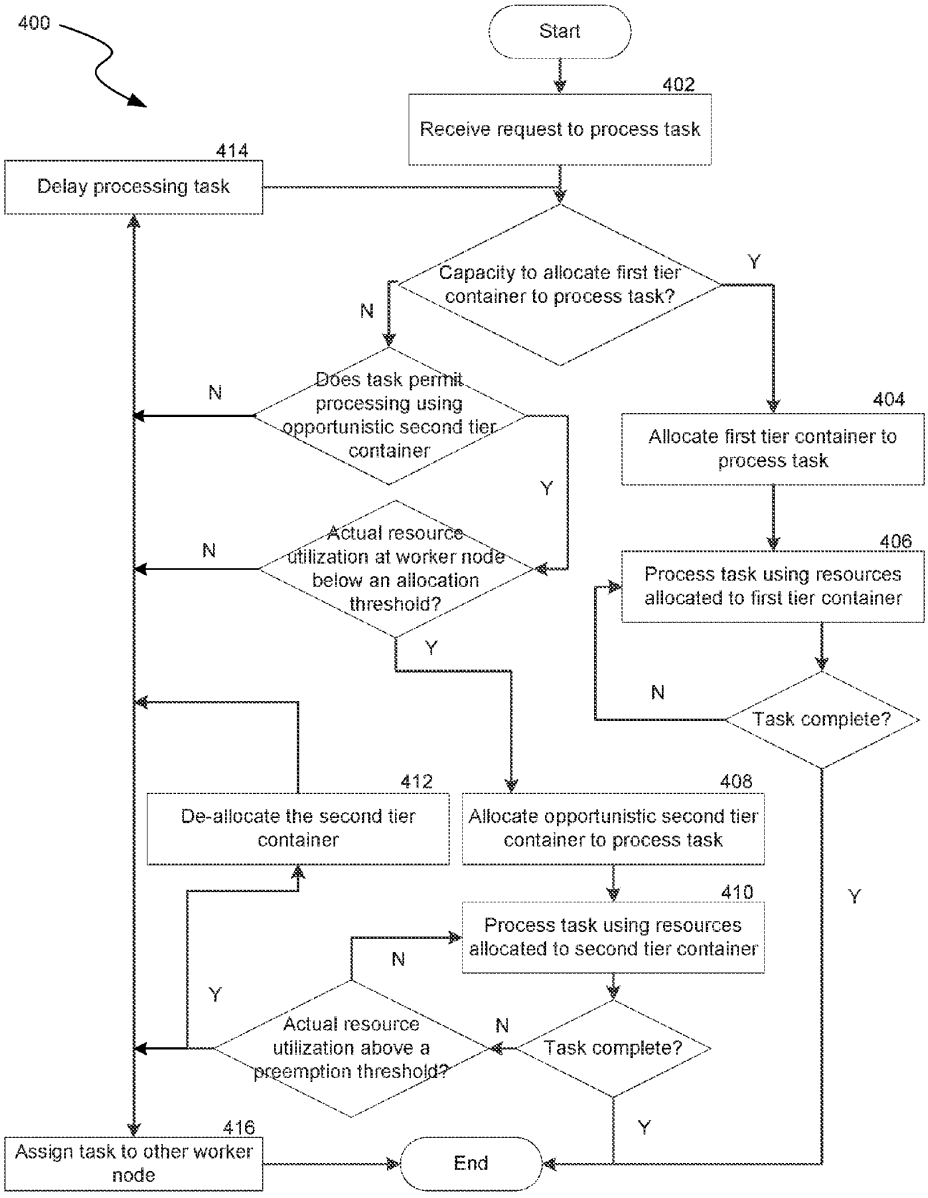
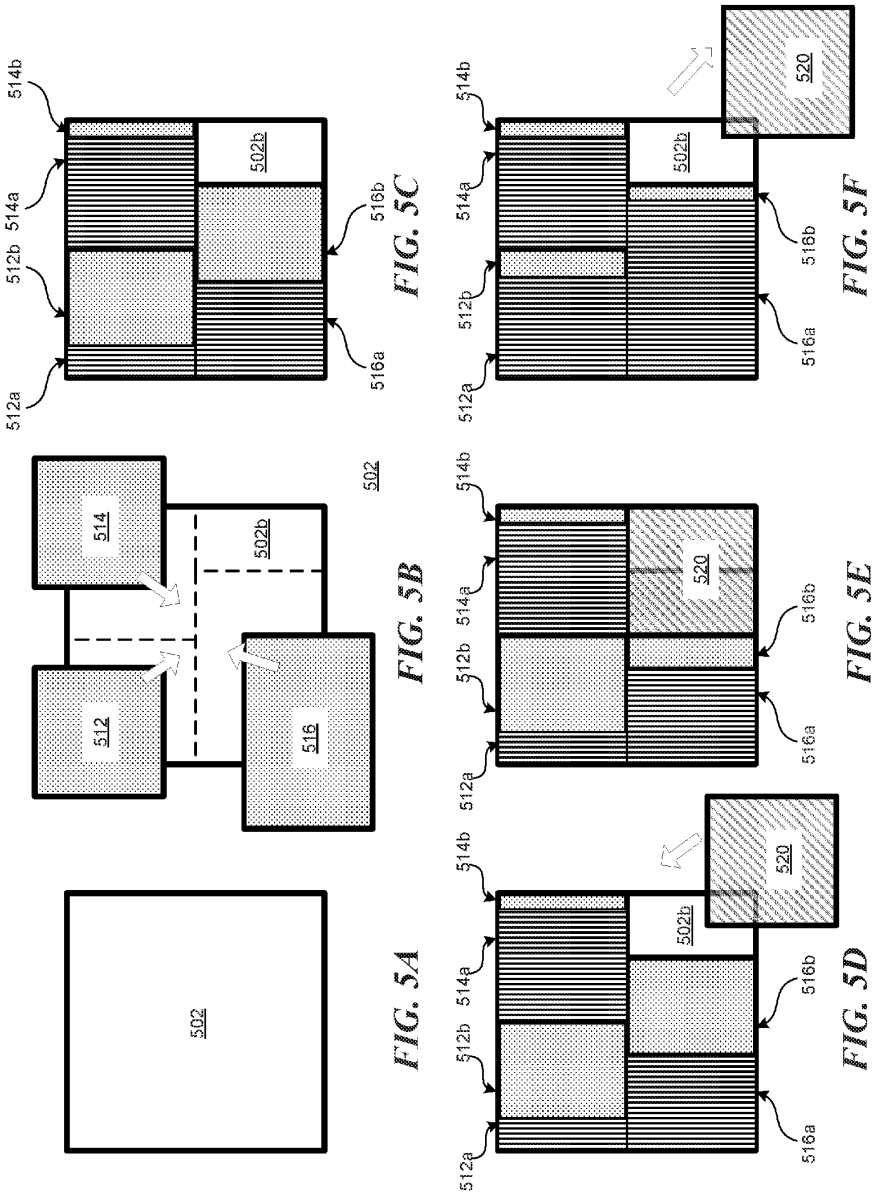


FIG. 4

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 199 of 411



ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 200 of 411

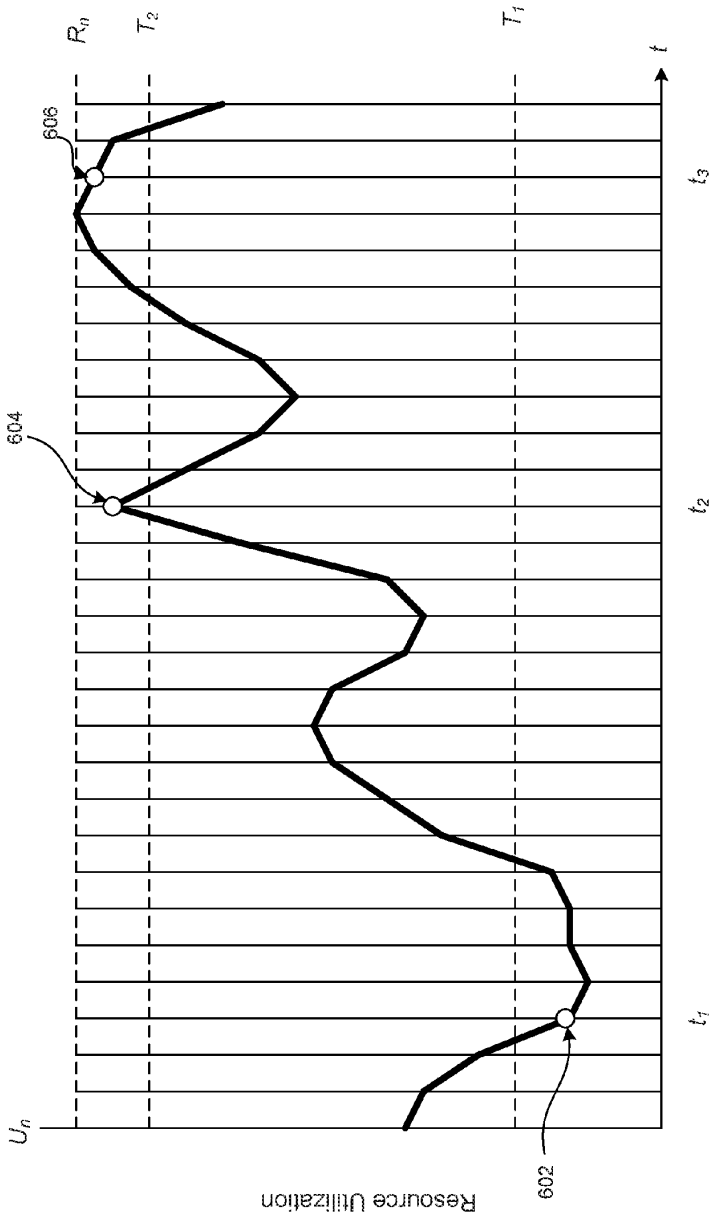


FIG. 6

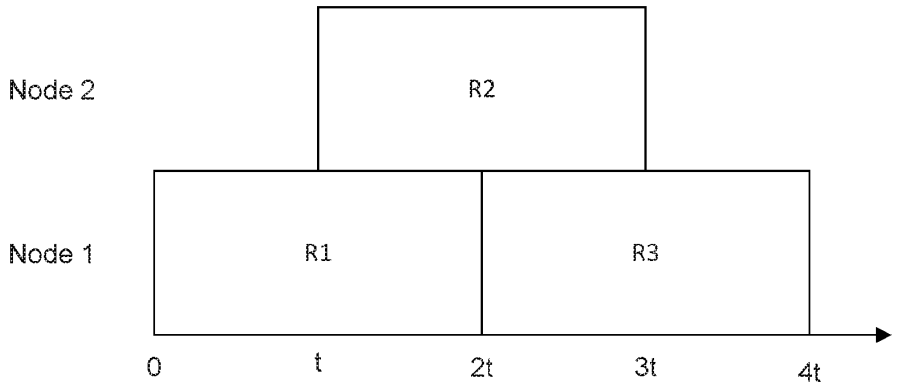


FIG. 7A

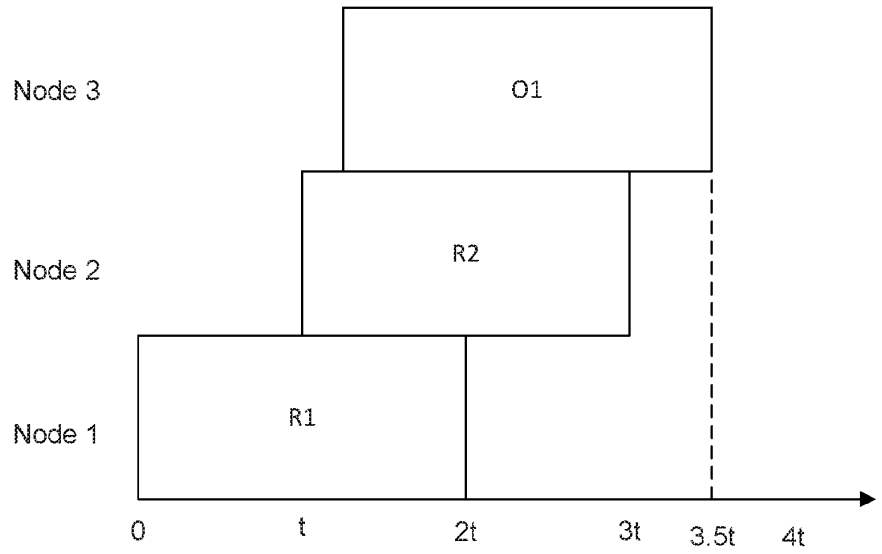


FIG. 7B

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 202 of 411

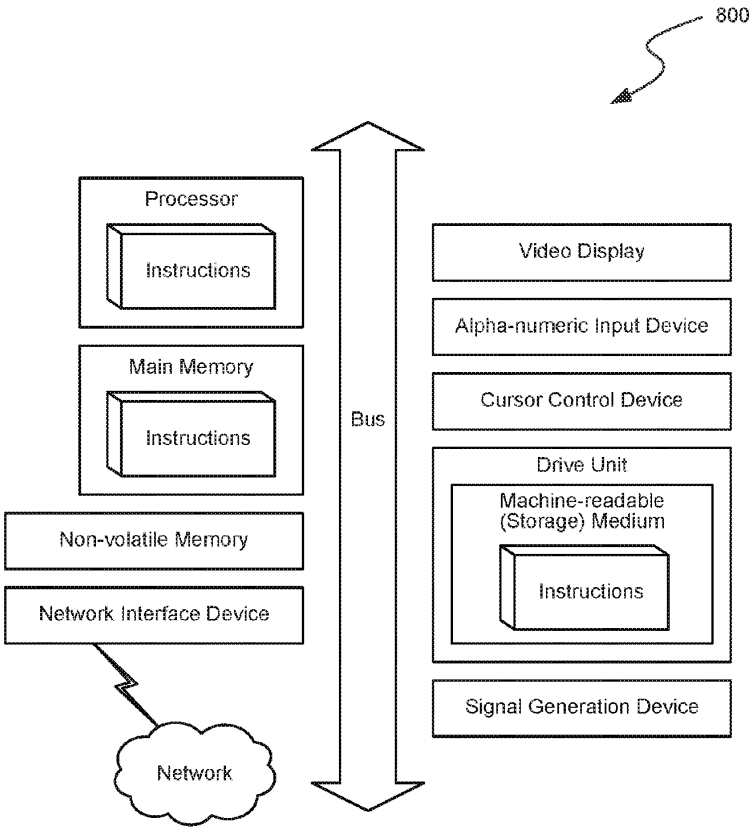


FIG. 8

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 203 of 411

UTILIZATION-AWARE RESOURCE SCHEDULING IN A DISTRIBUTED COMPUTING CLUSTER

CROSS-REFERENCE TO RELATED APPLICATION(S)

[0001] This application is entitled to the benefit of and/or the right of priority to U.S. Provisional Application No. 62/394,660, entitled "UTILIZATION-AWARE RESOURCE SCHEDULING" (Attorney Docket No. 68784-8024.US00), filed Sep. 14, 2016, which is hereby incorporated by reference in its entirety. This application is therefore entitled to an effective filing date of Sep. 14, 2016.

TECHNICAL FIELD

[0002] The present disclosure generally relates to distributed computing clusters, and more particularly to the allocation and management of computing resources in distributed computing clusters.

BACKGROUND

[0003] Collecting and analyzing increasingly large amounts of data is integral to the efficient operation of modern-day enterprises. Data-centric programming models like Apache Hadoop MapReduce or Apache Spark are commonly used for such data analyses tasks. Apache Hadoop project (hereinafter "Hadoop") is an open-source software framework for developing software for reliable, scalable and distributed processing of large data sets across clusters of commodity machines. Hadoop includes a distributed file system, known as Hadoop Distributed File System (HDFS). HDFS links together the file systems on local nodes to form a unified file system that spans an entire Hadoop cluster. Hadoop can also be supplemented by other Apache projects including Apache Hive (hereinafter "Hive") and Apache HBase (hereinafter "HBase"). Hive is a data warehouse infrastructure that provides data summarization and ad hoc querying. HBase is a scalable, distributed NoSQL (No Structured Query Language) database or data store that supports structured data storage for large tables.

[0004] MapReduce and Spark jobs typically include multiple tasks, each processing a partition of the overall input for the job. A cluster scheduler, like Apache Hadoop YARN or Apache Mesos allows sharing of cluster computing resources among several jobs, potentially from multiple users. Existing cluster schedulers (e.g. YARN) support a scheduling model based on resource requests. In other words, jobs submitted by users can include a request for certain resources (e.g. CPU, memory, etc.) needed to process the job. In turn, a cluster scheduler can allocate resources at nodes in a computer cluster when they become available. Such resource allocations are generally referred to as containers. The computing resources allocated within a given container are reserved exclusively for use within the given container and cannot be used by other containers, even if the allocated resources are not currently being utilized.

[0005] The amount of computing resources required to process a given task can be difficult to predict. It is inevitably difficult to accurately estimate the resource requirements of a job or its constituent tasks because: (i) resource usage of a task varies over time, and (ii) resource usage can vary across tasks of the same job based on the input they process. Users are expected to estimate and request the peak usage across all tasks to ensure job completion. This problem is further exacerbated by the fact that end-users can use convenience wrapper libraries like Apache Hive to create a majority of these jobs, and are consequently unaware of their characteristics. For these reasons, in practice, users end up using defaults, picking very conservative estimates of peak utilization (e.g. based on historical usage), or copying resource requirements from other work-flows that are known to work. The over-allocation of resources to process jobs and tasks leads to resource fragmentation and severe under-utilization of the computing resources in the cluster.

BRIEF DESCRIPTION OF THE DRAWINGS

[0006] The present embodiments are illustrated by way of example and are not intended to be limited by the figures of the accompanying drawings. In the drawings:

[0007] FIG. **1** shows a high-level diagram illustrating an example environment in which utilization-aware resource scheduling can be implemented;

[0008] FIG. **2** shows a block diagram illustrating example components of a unified distributed computing platform supporting utilization-aware resource scheduling;

[0009] FIG. **3**A shows a block diagram illustrating example components of a resource manager daemon installed at a master node in a distributed computing cluster;

[0010] FIG. **3**B shows a block diagram illustrating example components of a resource manager daemon installed at worker nodes in a distributed computing cluster;

[0011] FIG. **4** shows a flow diagram illustrating an example process for utilization-aware resource scheduling in a distributed computing environment;

[0012] FIGS. **5**A-SF show a series of conceptual graphical representations of resource allocation at a worker node using techniques described herein;

[0013] FIG. **6** shows a line graph that plots resource utilization at an example worker node over time;

[0014] FIG. **7**A shows a graph that illustrates a schedule for an example task that does not include opportunistic resource scheduling

[0015] FIG. **7**B shows a graph that illustrates a schedule for the example task of FIG. **7**A that does include opportunistic resource scheduling; and

[0016] FIG. **8** shows a diagrammatic representation of a machine in the example form of a computer system within which a set of instructions, for causing the machine to perform any one or more of the techniques discussed herein.

DETAILED DESCRIPTION

Overview

[0017] Introduced herein are techniques for utilization-aware cluster scheduling that address the aforementioned problem of underutilization of computing resources in a distributed computing cluster. These techniques are generally referred to herein as UBIS (utilization-based incremental scheduling) and can include scheduling techniques that consider both user-specified requests and the actual utilization of resources at a computing node, including computing resources previously allocated to a container. In an embodiment, once a computing cluster is fully allocated (i.e., no additional tasks can be scheduled based on un-allocated cluster resources), UBIS can opportunistically allocate computing resources not utilized by prior allocations. Such

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 204 of 411

opportunistic allocations are referred to herein as "opportunistic containers." For clarity, containers allocated according to known scheduling techniques (e.g. using existing YARN) are referred to herein as "regular containers." In contrast with so called "regular containers," opportunistic containers use slack in the cluster to improve cluster utilization and job throughput. Oversubscribing node and cluster resources in this manner, however, poses challenges. Oversubscription can become untenable when tasks simultaneously start using more resources, potentially leading to performance degradation, and even task failures. To address this problem, UBIS can preempt opportunistic containers to ease resource contention. However, preemptions limit throughput gains from opportunistic scheduling. To avoid unnecessary preemptions, individual jobs can opt out of opportunistic allocations. In some embodiments, cluster administrators can enable UBIS on a per-node basis to oversubscribe resources at the given node.

[0018] The potential for preemption of opportunistic containers in effect creates a hierarchy in container priority. The aforementioned regular containers can be considered as "first tier containers" that are guaranteed access to the computing resources allocated to the container. Accordingly, opportunistic containers can be considered as "second tier containers" in that they are subject to de-allocation to guarantee previously allocated computing resources to the first tier container if the resource utilization rises above a certain level that may impact access by the first tier container. The container hierarchy can of course include more than two levels. For example, the set of allocated regular containers may be organized into multiple sub-tiers (e.g. based on task priority and or time of allocation). Similarly, a set of allocated opportunistic containers may be organized into multiple sub-tiers (e.g. based on task priority and or time of allocation).

[0019] The aggressiveness of opportunistic scheduling can be controlled through a variable parameter. This parameter referred to herein as over-allocation parameter or threshold (T_{alloc}). UBIS can monitor and report resource utilization at a node and/or specific container to a scheduler. If the aggregate utilization is under a specified first threshold (e.g. based on T_{alloc}) the scheduler may allocate an opportunistic container to process a requested job or constituent task. UBIS can further allow limiting the extent of oversubscription at runtime through a second variable parameter referred to herein as a preemption parameter or threshold ($T_{preempt}$). UBS can preempt (i.e. de-allocate) previously allocated opportunistic containers if the aggregate resource utilization goes beyond a specified second threshold (e.g. based on $T_{preempt}$). Efficiency of resource utilization can thereby be controlled by selecting appropriate values for T_{alloc} and $T_{preeempt}$. The appropriate values will differ based on the processing requirements for a given environment, however in general a wider gap between the two thresholds will tend to reduce the likelihood of a node's resource usage actually going over the second threshold. This reduces the likelihood of preemption, while at the same time reducing opportunistic scheduling. The optimal values for these two thresholds for a cluster depend on the workload and resource capacities. For example, while most batch-processing jobs can tolerate preemptions in lieu of potential throughput gains, certain latency-sensitive applications might not be able tolerate preemptions. As will be explained later, these jobs can opt out of opportunistic allocations.

[0020] For illustrative purposes, embodiments are described in herein in the context a Hadoop-based distributed computing systems. For example, UBIS can be implemented in or for use with Apache YARN. While these described embodiments are useful for illustrative purposes, they are not to be construed as limiting. For example, while YARN currently supports allocation of containers based on CPU and memory resources, UBIS can be implemented for the allocation of other computing resources including, but not limited to, disk storage, network, GPU (graphical processing), bus, input/output (I/O) resources, etc. Further certain terms used herein such as "node," "job," "task," "workflow," "query," "resource," "container," etc., may have specific definitions in the context of a Hadoop-based system. Again, these should not be construed as limiting the scope of the presently described innovations. The techniques described herein (generally referred to as UBIS) can be applied to the processing of any type of data at any type of distributed system of computing devices (physical and/or virtual).

Distributed Computing Environment

[0021] FIG. **1** shows a high-level diagram illustrating an example environment **100** in which UBIS may be implemented. As previously mentioned, the example environment **100** is shown for illustrative purposes in the context of Hadoop, however this is not to be construed as limiting. Further, whether implemented in Hadoop or not, more or fewer components than as shown in FIG. **1** may be included while remaining within the scope of the present teachings. The example environment **100** includes a plurality of data nodes **124***a-c* that comprise cluster of worker nodes in in communication (e.g. via computer network) with each other and one or more master nodes. Some of the data nodes **124***a-c* may run just HDFS, while others may run HBase region servers **126***a-c*.

[0022] The environment **100** includes a client **104** such as Java Database Connectivity (JDBC) client, Open Database Connectivity (ODBC) client, and the like that provides API and other tools for connecting and/or accessing a Hadoop distributed computing cluster. SQL applications **102** such as Hue, provide a an interface through which users can run queries or jobs, browse the HDFS, create workflows and the like. The environment **100** also includes a command line interface **106** for issuing queries. In one embodiment, the client **104**, the SQL application **102** and the command line interface **106**, each or together may be commonly referred to as a client.

[0023] The environment **100** includes a central resource manager **108** and in some embodiments a secondary resource manager **110**. A "resource" in this context may be generally understood to include processing capabilities of the cluster nodes in the distributed computing environment. For example, resources may include processing units (e.g. CPU cores), memory, network bandwidth, and storage space. Resources may be abstracted into units called containers (e.g. containers **122***a-c*) that may include one or more of the previously mentioned resources. The central resource manager **108** is a general resource manager configured to manage and arbitrate resources among applications in the system. Communicating with node managers **118***a-c* which act as the agents at each node, the central resource manager **108** may allocate and schedule resources available at the various nodes based on the available resources reported

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 205 of 411

from each node manager **108**. In other words, queueing processing tasks until resources are made available. In Hadoop, this basic system of negotiation between a central resource manager **108** and node managers **118***a-c* is sometimes referred to as Yet Another Resource Negotiator (YARN). As previously discussed, a central resource manager **108** implementing known resource management techniques (such as in YARN) is limited in its ability to prevent over-allocation and wastage of computing resources. To address this limitation, environment **100** may include a second level resource manager such as a UBIS engine **110** capable of implementing novel techniques for utilization-aware resource management as discussed in more detail herein. UBIS engine **110** is shown in FIG. **1** as discrete module separate from central resource manager **108**, however it shall be understood that this shows an non-limiting example for illustrative purposes. According to some embodiments, the functionality of the UBIS engine **110** may be integrated into the central resource manager **108**. In other words, the novel methods and systems described herein may be implemented on top of an existing resource management architecture (e.g. YARN), or may be integrated into a new resource management system, according to some embodiments.

[0024] In some cases, YARN can implement one of several types of schedulers, some of which allow for user input to configure the queuing of jobs, for example to effectuate relative priority among jobs. For example, YARN can implement a fair scheduler that allows an admin to assign weights to certain job queues. The fair scheduler then computes a "fairshare" for each queue/application in a hierarchy based on these weights and priorities the allocation of computing resources based on the calculated "fairshare." In some embodiments, UBIS-based techniques described herein can be implemented within the scheduler in a given YARN deployment.

[0025] Elements depicted in the distributed computing environment of FIG. **1** may be implemented across one or more physical and/or virtual computing devices. These one or more computing devices comprise any combination of hardware and software configured to implement the various logical components described herein. For example, the one or more computing devices may include one or more memories that store instructions for implementing the various components described herein, one or more hardware processors configured to execute the instructions stored in the one or more memories, and various data repositories in the one or more memories for storing data structures utilized and manipulated by the various components. Additional detail regarding components of the one or more computing devices is described with respect to FIG. **8**.

[0026] In some embodiments, the computing devices (real or virtual) implementing the elements depicted in FIG. **1** may be communicatively coupled via one or more computer networks. The one or more computer networks can include, for example, a wired or wireless local area network (LAN), a wired or wireless wide area network (WAN), a cellular data network, or any combination or variation thereof. Further, the one or more networks can include open networks (e.g. the Internet) and/or private network (e.g. an intranet and/or the extranet). For example, the Internet can provide file transfer, remote log in, email, news, RSS, cloud-based services, and other services through any known or convenient protocol, such as, but is not limited to the TCP/IP protocol, Open System Interconnections (OSI), FTP, UPnP, etc.

[0027] The one or more networks can be any collection of distinct networks operating wholly or partially in conjunction to provide connectivity to the computing devices implementing the elements of FIG. **1** and may appear as one or more networks to the serviced devices. In one embodiment, communications to and from computing devices can be achieved by, an open network, such as the Internet, or a private network, such as an intranet and/or the extranet. In one embodiment, communications can be achieved by a secure communications protocol, such as secure sockets layer (SSL), or transport layer security (TLS).

[0028] FIG. **2** shows a block diagram illustrating example components of a unified distributed computing platform **212** supporting batch-oriented and real-time, ad hoc queries. The unified Hadoop platform **212** supports distributed processing and distributed storage. The unified Hadoop platform **212** includes a user interface **214**, storage **220** and metadata **222** components. The user interface **214** includes Hive interfaces such as ODBC driver, JDBC driver, Hue Beeswax, and the like. The user interface **214** also includes SQL support. Via the user interface **214**, queries can be issued, data can be read from or written to storage **220**, etc. Through the interface, an administrator user can set policies for resource allocation in processing queries. These policies may inform resource allocation by a central resource manager **108** or UBIS engine **110**. The storage **220** includes HDFS and/or HBase storage. The HDFS may support various file formats, including but not limited to: text file, sequence file, RC file, Avro, and the like. Various compression codecs including snappy, gzip, deflate, bzip, and the like may also be supported. The metadata **222** may include, for example, information about tables, their partitions, columns, types, table/block locations, and the like. The metadata **222** may leverage existing Hive metastore, which includes a mapping of HBase tables, predicates on row key columns mapped into start/stop row, predicates on other columns mapped into single column value filters, and the like.

[0029] According to some embodiments, a Hadoop platform uses a batch-oriented query engine (e.g., MapReduce) for batch processing **216** of data. The batch processing capability of MapReduce is complemented by a real-time access component **218** in the unified Hadoop platform **212**. The real-time access component **218** allows real-time, ad hoc SQL queries to be performed directly on the unified storage **220** via a distributed low latency (LL) query engine that is optimized for low-latency. The real-time access component **218** can thus support both queries and analytics on big data.

[0030] FIG. **3**A shows a block diagram illustrating example components of a resource manager daemon installed at a master node in a distributed computing cluster. The resource manager daemon shown in FIG. **3**A may include components of a central resource manager **108** and/or UBIS engine **110** as shown in FIG. **1**. Further, the conceptualized diagram of a resource manager daemon shown in FIG. **3**A is an illustrative example according to one embodiment and is not to be construed as limiting. A similar resource manager include more or fewer components, organized in various configurations, while remaining within the scope of the present disclosure.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 206 of 411

[0031] According to one embodiment, a resource manager daemon includes a client services module **310***a*, an administrator services module **320***a*, an application manager **330***a*, a scheduler **340***a*, and a resource tracker **350***a*.

[0032] The client services module **310***a* provides the interface between the client (e.g., client **102**/**104**) and the resource manager (e.g. resource manager **108**/**110**). For example, the client services module **310***a* may handle communications (e.g., in the form of remote procedure calls) for requested operations (e.g. a query) to be performed on the cluster.

[0033] The administrator services module **320***a* provides the interface between the client (e.g., client **102**/**104**) and the resource manager (e.g. resource manager **108**/**110**) for administrator services. Further, the administrator services module **320***a* can prioritize operations requests from administrators over normal users. An administrator can set resource management policies to be followed by the scheduler **340***a* via the administrator services module **320***a*.

[0034] The application manager **330***a* is responsible for maintaining a collection of submitted "applications" or "jobs." In this context, the terms "jobs" and "applications" may be used interchangeably, however an application may also include a set of jobs. For example, a MapReduce job may be considered an "application" or a software application such as a Hive instance may be considered an "application" comprising a number of MapReduce jobs at any given time. Further, different instances or accounts for the same software application may be considered distinct applications by the application manager **330***a*. For example, in an enterprise setting, various departments may have access to Hive-based software for processing queries using a Hadoop Cluster. Each department's use of the software may be separately managed as an "application." The application manager **330***a* may also include an application master service responsible for communicating with all the application masters **320***b* at the slave nodes. The application master service can register new application masters **320***b*, terminate and/or unregister requests from finishing application masters **320***b*, and obtain container allocation or de-allocation requests from running application masters **320***b*.

[0035] The scheduler **340***a* is responsible for allocating resources to the various applications subject to constraints queues and policies set by an administrator user. Scheduling is performed based on scheduling algorithms that take into consideration the resource requirements (e.g. memory, processing, data storage, network bandwidth, etc.) of the submitted applications, the administrator policy requirements, and other constraints.

[0036] The resource tracker **350***a* responds to remote procedure calls from the worker nodes. It monitors available resources at the nodes, by receiving status updates from the worker nodes. The resource tracker **350***a* may also decommission resources at nodes if it does not receive status updates indicating that the node is operational. The resource tracker **350***a* maintains a list of active and decommissioned nodes and feeds this information to the scheduler **340***a* to aid in resource allocation.

[0037] FIG. **3**B shows a block diagram illustrating example components of a resource manager daemon installed at worker nodes in a distributed computing cluster, according to some embodiments. The conceptualized diagram of a resource manager daemon shown in FIG. **3**B is an illustrative example according to one embodiment and is not to be construed as limiting. A similar resource manager include more or fewer components, organized in various configurations, while remaining within the scope of the present disclosure.

[0038] According to one embodiment, a resource manager daemon installed at the worker nodes includes a node manager **310***b*, and one or more application masters **320***b*. The node manager **310***b* depicted in FIG. **3**B may be the same as or part of node managers **118***a-c* depicted in FIG. **1**. Resources at the worker nodes are abstracted to one or more containers **330***b*.

[0039] The node manager **310***b* is a per-node agent installed on each of the worker nodes in the cluster. The node manager **310***b* includes a node status updater **312***b* that registers with the resource manager and broadcasts the status of the node including the status of available resources (e.g. containers **330***b*) at the node. For example, the node status updater **312***b* of the node manager **310***b* can periodically send heartbeat signals to the resource manager **108**/**110** that include liveness and the status of containers **330***b* allocated at the worker node. Status updates may include information about new allocated containers, completed containers, unavailable containers, etc. The node manager **310***b* also includes a container manager **314***b*. The node manager **310***b* can monitor the resource utilization of individual containers, for example, by using a proc filesystem on Unix-based operating systems. Alternatively, in some embodiments, performance monitoring may be improved through more advanced operating system libraries like cgroups in Linux. In some embodiments, the node manager **310***b* can be augmented to also calculate the aggregate resource utilization across all containers in the worker node. This aggregate information can then be sent to the resource manger **108**/**110** at the master node to inform UBIS scheduling. In some embodiments, signals sent by the node manager **310***b* at a worker node to the resource manger **108**/**110** (e.g. as periodic heartbeats) at the master node can include one or more of (i) the aggregated actual resource utilization at the worker node, and/or (ii) an indication of the willingness of the worker node to accept over-allocation. For example, as previously mentioned, this indicator may be in the form of a parameter or threshold T_{alloc}. As previously mentioned, in some embodiment T_{alloc} can be based at least in part on the actual resource utilization at the worker node. This additional information will typically add little overhead to an existing node heartbeat payload.

[0040] The node container manager **314***b* accepts requests from application masters **320***b* for resources, for example, to start running new containers **330***b* or stop running old ones, launches and monitors containers **330***b* at the node, closes containers and/or cleans up processes running in containers in response to requests from the resource manager **108**/**110**, and monitors the containers **314***b* for resource utilization while processes are running. According to some embodiments, if a container **330***b* exceeds its resource allocation, the node container manager **314***b* can kill the errant container so that it does not adversely affect other containers running on the same node. The node manager **310***b* also includes a container executor **316***b* that interacts with underlying operating systems to place files and directories and to securely launch and clean up processes running on containers.

[0041] Application masters **320***b* that are specific to each application running on the cluster may be instantiated at

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 207 of 411

each worker node. Application masters **320***b* are responsible for requesting resources from the resource manager **108**/**110**, working with node managers **310***b* to manage and monitor the containers **330***b* allocated by the resource manager **108**/**110**.

Optimizing Resource Utilization

[0042] UBIS can be implemented to improve effective resource utilization in a distributed computing cluster. Improvements in effective resource utilization can positively impact metrics that directly impact end users such as makespan (i.e., processing duration) for individual jobs and workloads comprising sets of jobs. Generally speaking, a workload can be viewed as a general directed acyclic graph (DAG) of jobs, where each job comprises several tasks. End-users typically submit workloads and not individual jobs. A single job is the basic unit of a workload, and makespan for a job is simply the job duration. In other words, in some embodiments UBIS can be implemented with a goal of minimizing the overall workspan of submitted jobs and workloads. In many cases this involves striking a balance between over-allocation and guaranteeing allocated cluster resources due to the way in which tasks are processed in a given distributed cluster.

[0043] Consider a cluster with R resources (where R is a multidimensional vector, with dimensions representing CPU, memory, network, disk, and potentially other resources), and a workload comprising of N jobs with a total of n tasks. A user-specified maximum resource requirement for a given task T_i is denoted as R_i. The maximum resource requirement R_i for the given task T_i can be computed by maximizing along each dimension over the execution of the task. At any given time, a cluster scheduler using existing techniques schedules the largest subset of tasks that fit on the cluster as $G=\{T_i|\Sigma R_i \leq R\}$. In other words, adding one more task to the set would yield $\Sigma R_i > R$. The actual resource utilization of a given task T_i running on the cluster is represented as U_i. To successfully process the task T the actual resource utilization U_i should be less than the maximum resource requirement R_i specified for the task. Therefore, the slack in the cluster due to underutilization can be represented as $R^*=\Sigma R_i - \Sigma U_i$.

[0044] In some embodiments, UBIS can be implemented to schedule opportunistic task that fit within this available slack R* in the cluster. In other words, a cluster scheduler implementing UBIS (in some embodiments) could schedule the largest subset of tasks that fit within this available slack R* as $0=\{T_i|\Sigma R_i \leq R^*\}$. However, utilizing all available resources, including previously allocated slack, at any given time can lead to resource contention between competing tasks due to temporal variations in resource utilization. Such resource contention can, in some cases, lead to failure of certain tasks. Depending on task/job/workload characteristics, failure of one task can in some cases negatively impact the makespan of a corresponding job and/or workload. In other words, maximum overall resource utilization does not necessarily yield the optimal effective resource utilization. The associated loss in performance can be represented as Δt_i, where t_i is the duration of task T_i. In cases of extreme contention between tasks for resources, a select number of tasks (p) may need to be preempted (i.e. suspended and/or de-allocated) to ensure that oversubscription in the cluster remains tenable. Accordingly, the following observations can be made in such a scenario:

[0045] Makespan is inversely related to the number of tasks run in parallel.

[0046] Makespan grows with individual task duration $(t_i+\Delta t_i)$.

[0047] The increase in task duration (Δt_i) due to resource contention grows with number of tasks run in parallel. For an unsustainable number of scheduled tasks, the overhead may cause significant thrashing.

[0048] Makespan grows with number of preemptions (p).

[0049] The optimal makespan will therefore correspond to the largest value of utilized slack for which resource contention is manageable (i.e. where Δt_i and p are minimized). This will depend on the workload being processed as well as the cluster resources.

Example UBIS Process—Scheduling Opportunistic Containers

[0050] FIG. **4** is a flow chart describing an example UBIS process **400** that includes the scheduling of opportunistic containers. For clarity and illustrative purposes, the steps of process **400** are described with respect to the Hadoop-based computing environment illustrated in FIGS. **1**-**3**B. However, a person having ordinary skill will recognize that process **400** may be carried out in a distributed computing cluster that does not necessarily conform with the architectural components illustrated in FIGS. **1**-**3**B. Further process **400** shown in FIG. **4** is an example of a process that incorporates techniques for UBIS. Other embodiments of UBIS may implement processes that order the steps differently and/or include more or fewer steps than as illustrated in FIG. **4**.

[0051] The example process **400** begins at step **402** with receiving a request to process a task in a distributed computing cluster. The task may be part of a job or workload (set of jobs) submitted for processing in a distributed computing cluster. Jobs and/or workloads may be submitted by end-users (e.g. via clients **102**/**104**) and are received at a master node (e.g. at a resource manager **108**/**110** associated with the cluster. The jobs and/or workloads may include user-specified resource requirements.

[0052] As previously mentioned, in certain embodiments the resource manager **108**/**110** receives periodic heartbeat signals from the node mangers **310***b* at the worker nodes. When a node heartbeats, the scheduler **340***a* at the resource manager **108**/**110** updates its book-keeping for any containers completed since the last heartbeat, and any resource capacity updates. If a worker node includes unallocated capacity to process the task (i.e. based on the requested resources) the scheduler **340***a* at step **404** allocates or causes the allocation of one or more regular (i.e. first tier) containers to process the requested task.

[0053] FIGS. **5**A-**5**F serve to further illustrate portions of the process **400** described with respect to FIG. **4**. Specifically, FIGS. **5**A-**5**F include a series of graphical representations of resource allocation at a particular worker node at different times. It shall be understood that the graphical representations are conceptual and do not necessarily correspond with how resources are actually arranged in practice. Further, the representations in FIGS. **5**A-**5**F are exemplary and are not intended to limit the present teachings to particular processes for resource allocation. Element **502** shown in FIG. **5**A conceptually represents a set of unallocated resources (e.g., at a particular worker node). Again, the resources may be multi-dimensional and include processing,

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 208 of 411

memory, storage, network etc. To process requested tasks, a scheduler can allocate one or more regular first tier containers as represented by elements **512**, **514**, and **516** shown in FIG. **5**B. The containers may be sized according to task resource requirements or may be standardized. Because allocated resources are guaranteed to each regular container, the elements **512**, **514**, and **516** are represented in FIG. **5**B to fit within the span of available resources represented by element **502** while not overlapping each other.

[0054] Returning to FIG. **4**, the task is then processed at step **406** using the resources allocated to the one or more regular containers. As previously mentioned, regular first tier containers include allocated resources that are guaranteed. In other words, the task will continue to process at step **406** without any risk of de-allocation of the regular container. In some embodiments where a number of requested tasks are pending, the scheduler **340***a* may iterate through the waiting tasks (e.g. in an order determined by fairness constraints and queue weights) to check if the worker node meets the resource size and placement requirements (say, for data locality) for the task. The scheduler **340***a* continues to allocate regular containers at a particular worker node, as long as it has enough resources to meet a job's pending request.

[0055] If, however, a particular worker node or all of the nodes in the cluster do not have enough unallocated resources available to allocate a regular first tier container (i.e. the scheduler **340***a* has completed allocating regular containers), process **400** continues with determining whether to opportunistically allocate unused (but previously allocated) resources to process the task. As will be described, in some embodiments, tasks may opt out of the opportunistic resources scheduling by UBS. For example, some tasks may not be able to tolerate any risk of preemption by higher tiered containers. Accordingly, in some embodiments, determining whether to allocate an opportunistic container includes first determining whether the requested task permits processing using opportunistic containers. If the task does not permit processing using an opportunistic container, the requested task may at step **414** be delayed (e.g. placed back into a job queue). Alternatively, if resource availability has opened up at any of the other worker nodes in the cluster the scheduler **340***a* may opt to re-assign the requested task to another worker node and allocate a first tier container at that node to process the task.

[0056] Assuming that the requested task permits, there may be an opportunity to allocate a second tier opportunistic container to process the task. Consider again the graphical representations of resource allocation shown at FIGS. **5**A-**5**F. During processing of tasks using previously allocated regular containers, some of the resources in the regular containers may be utilized, however at any given time some of the allocated resources may remain unutilized. For example, FIG. **5**C shows the regular containers (represented by elements **512**, **514**, and **516**) broken down into utilized portions and unutilized portions. Specifically, at a given time, tasks processed at regular containers represented by elements **512**, **514**, **516** may only utilize the resources represented by elements **512***a*, **514***a*, **516***a* (respectively) leaving the allocated resources represented by elements **512***b*, **514***b*, **516***b* unutilized. In other words, the resource slack at the worker node represented in FIGS. **5**A-**5**F at any given time can be represented as the sum of the unutilized allocated resources **512***b*, **514***b*, **516***b*, and the unallocated resources **502***b*.

[0057] As soon as resource slack at a worker node develops there may be an opportunity to opportunistically allocate some or all of that slack. As previously mentioned, the maximum quantity of resources that may be opportunistically allocated at any given time is equal to the total resource slack at that given time. However, due to temporal fluctuations in resource utilization, opportunistically allocating all of the slack and/or opportunistically allocating as soon as slack develops can lead to contention. Instead, the scheduler **340***a* can determine whether to allocate an opportunistic second tier container based on whether actual resource utilization at a particular node satisfies a condition for opportunistic scheduling. For example, in an embodiment, the scheduler **340***a* may only allocate an opportunistic second tier container if the actual resource utilization is below an allocation threshold. In other words, the scheduler **340***a* may only allocate an opportunistic container to process a task if the worker node has available unused resources to process the task (according to the task's requirements), wherein the quantity of available resources takes into account the worker node's willingness to take on opportunistic containers (e.g. the variable allocation parameter T_{alloc}). For example, FIG. **6** shows a line graph (not to any particular scale) that plots resource utilization U_n at an example worker node over time, t. As shown in FIG. **6**, an opportunistic second container may be allocated at t_1 (represented by point **602** along the graph) as resource utilization falls below a first threshold T_1. Again, the first threshold T_1 may at least in part be based on a variable allocation parameter T_{alloc} (e.g., $T_1=T_{alloc}\times R_n$, wherein R_n is the resource capacity at the worker node and T_{alloc} has a value between 0 and 1). Note that the line graph of FIG. **6** is an example provided for illustrative purposes and is not to be construed as limiting. Further, the placement of thresholds T_1 and T_2 along the y-axis of the line graph are not necessarily to scale relative to each other or to the overall resource capacity R_n of the node.

[0058] In an embodiment, a system implementing UBIS identifies resource slack at each worker node from unused resources of previously allocated running containers. For this, UBIS can actively monitor resource usage of each container at the worker nodes and of the worker nodes themselves. For example, the node managers **310***b* at worker nodes can heartbeat liveness information to the resource manager **108**/**110** at the master node periodically. As previously mentioned this heartbeat information can further include utilization information as well as an indicator of the worker node's willingness for over subscription (e.g. the variable over-subscription parameter, T_{alloc}). A scheduler **340***a* can allocate opportunistic containers at a given worker node if the actual resource utilization U_n at the worker node hits or falls below a threshold equal to $T_{alloc}\times R_n$, where R_n is the worker node's resource capacity for running containers and T_{alloc} has a value between 0 and 1. Note that in some embodiments, the actual resource utilization U_n may need to just hit the threshold of $T_{alloc}\times R_n$, while other embodiments may require that the actual resource utilization U_n fall below this threshold. As mentioned, in some embodiments, the resource manager **108**/**110** may receive periodic heartbeats from the node managers **310***b* at worker nodes. Accordingly in some embodiments, this determination may be made by

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 209 of 411

the resource manger **108**/**110** each time a heartbeat signal is received (e.g., every 1 second). In some embodiments, to remain conservative and avoid over allocation due to momentary drops in actual utilization, the scheduler **340***a* of the resource manager **108**/**110** may require that actual resource utilization U_n remain at or below the threshold of $T_{alloc} \times R_n$ for a certain period of time before determining that conditions are met for opportunistic scheduling. For example, in an embodiment, the scheduler **340***a* may wait until the actual resource utilization U_n has remained at or below the threshold of $T_{alloc} \times R_n$ for a predetermined number of periodic heartbeats (e.g. 10 heartbeats). The resources available at a worker node for opportunistic scheduling at any given time can similarly be expressed as $(T_{alloc} \times R_n) - U_n$. Therefore, in some embodiments, a requested task with a user-specified maximum resource requirement R_i can be opportunistically scheduled if $R_i \leq (T_{alloc} \times R_n) - U_n$ at the worker node. Analogizing to the graphical representations resource allocation in FIGS. **5**A-**5**F, an opportunistic container as represented by element **520** can be allocated to process a requested task if its area is less than or equal to the total area of the element **502** multiplied by T_{alloc} minus the sum of the areas of **512***a*, **514***a*, and **516***a* (e.g. as shown in FIGS. **5**D-SE).

[0059] In some embodiments, the variable allocation parameter T_{alloc} may be set at a particular value (e.g., between 0 and 1). The T_{alloc} parameter can also be generalized to a vector, with dimensions representing different resource types. While the parameter T_{alloc} may be cluster-wide, in some embodiments it may be specific to a worker node given the characteristics and/or capabilities of the worker node. For example, in practical implementations, worker nodes may have different hardware, storage hot-spots, and other deployed software. Further, in some embodiments, the parameter T_{alloc} may be job specific. Some job types may be more amendable to opportunistic scheduling than others.

[0060] The optimal values for T_{alloc} will depend on worker node resources and the workload scheduled on the node. A high value for T_{alloc} indicating a high willingness to over-subscribe resources will lead to a high number of opportunistic containers. This is sustainable if the utilization of running containers remains steady. However, major fluctuations in utilization can lead to severe contention and pre-emptions. In practice, optimal values for T_{alloc} may be arrived at through trial-and-error. For example, a cluster administrator may run a series of predictable workloads and track the overall sensitivity to varying values for T_{alloc}, whether implemented cluster wide or at specific nodes. Tools can be provided (e.g. via a user interface **214** of a unified distributed computing platform **212**) that can enable a cluster administrator to fine tune values for T_{alloc} (cluster wide, on per node basis, on a per job-type basis, etc.). To inform these adjustments by the cluster administrator, real-time and/or historical data regarding cluster utilization and performance can be provided as well.

[0061] Setting values for T_{alloc} using trial-and-error may work in some implementations, however may be impractical in others. Accordingly, in some embodiments, values for T_{alloc} (cluster wide, on per node basis, on a per job-type basis, etc.) can be set automatically using certain optimization techniques. For example, optimal value for T_{alloc} may be determined through a feedback loop from past runs, for example by using gradient-descent.

[0062] Returning to FIG. **4**, assuming the conditions for opportunistic allocation are satisfied, the scheduler **340***a* may at step **408** allocate an opportunistic second tier container and at step **410** begin processing the requested task using the resources of the opportunistic second tier container. If the conditions for opportunistic scheduling are not met, the requested task may at step **414** be delayed (e.g. placed back into a job queue). Alternatively, if resource availability has opened up at any of the other worker nodes in the cluster the scheduler **340***a* may opt to re-assign the requested task to another worker node to process the task.

[0063] In some embodiments, the above described process of scheduling opportunistic containers can be performed as long as the conditions for opportunistic scheduling (based on resource utilization) are satisfied at a particular node. Further, as previously alluded to, in some cases, subsequently allocated opportunistic containers may be of a different tier than previously scheduled opportunistic containers. For example, although not shown in FIG. **4**, in an embodiment process **400** may continue with the scheduler **340***a* allocating an opportunistic third tier container at the particular worker node to process another task in response to determining that the actual computing resource utilization at the particular worker node is below an allocation threshold. In this example, the opportunistic third tier container may include underutilized computing resources previously allocated to regular first tier container and/or opportunistic second tier containers. Due to its lower tier status, the opportunistic third tier container would be subject to de-allocation to guarantee the computing resources to the regular first tier container and/or the opportunistic second tier container if the actual computing resource utilization at the particular worker node rises above a particular threshold.

Example UBIS Process—Avoiding Adverse Effects of Opportunistic Containers

[0064] The addition of opportunistic containers for improved utilization can lead to contention, sometimes severe enough to render a worker node unusable. In such situations, the node manager **310***b* (or even the operating system) may not be able to take corrective action to prevent processing failures. To avoid these adverse conditions, task executing in opportunistic second tier containers can be subject to preemption to guarantee previously allocated resources to regular containers before processing task enter into an unrecoverable state. In some cases this may involve de-allocating an opportunistic container before the task has completed.

[0065] Accordingly, while processing a task using an allocated opportunistic container at step **410**, process **400** may further include determining whether to preempt the task if necessary to guarantee resources to regular first tier containers. For example, in an embodiment the scheduler **340***a* may opt to preempt the task running in an opportunistic second tier container if the actual resource utilization rises above a preemption threshold. Similar to the allocation threshold, the preemption threshold may, at least in part, be based on a variable preemption parameter $T_{preempt}$. For example, in an embodiment, a task running in an opportunistic container may be preempted if actual utilization U_n hits or rises above $T_{preempt} \times R_n$, where R_n is the worker node's resource capacity for running containers and $T_{preempt}$ has a value between 0 and 1. Similarly, in some embodi-

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 210 of 411

ments, a task with user-specified resource requirement R_t may be preempted if $R_t \geq (T_{preempt} \times R_n) - U_n$.

[0066] In some embodiments, to remain conservative and avoid unnecessarily preempting certain tasks due to momentary spikes in actual utilization, the scheduler **340***a* of the resource manager **108**/**110** may require that actual resource utilization U_n remain at or above the threshold of $T_{preempt} \times R_n$ for a certain period of time before determining that conditions are met for preempting the executing task. For example, in an embodiment, the scheduler **340***a* may wait until the actual resource utilization UL has remained at or above the threshold of $T_{preempt} \times R_n$ for a predetermined number of periodic heartbeats (e.g. 3 heartbeats). This is illustrated in FIG. **6** in which the momentary spike in actual resource utilization over a second threshold T_2 at time t_2 (shown at point **604**) may not trigger preemption, while sustained utilization above T_2 at time t (shown at point **606**) does trigger preemption.

[0067] Preemption of an opportunistic second tier container may involve different steps. For example, in an embodiment of process **400** described with reference to FIG. **4**, the opportunistic second tier container may be de-allocated at step **412** resulting in termination of any task processing in the opportunistic container. Once the opportunistic container is de-allocated, the requested task may at step **414** be delayed (e.g. placed back into a job queue). Consider again the graphical representations resource allocation in FIGS. **5**A-**5**F. As shown in FIG. **5**E tasks may continue to execute in an opportunistic container (as represented by element **520**) as long as the resource utilization conditions permit. However, increases in overall resource utilization may trigger preemption of these tasks. For example, FIG. **5**F shows a condition in which resource utilization (represented by elements **512***a*, **514***a*, and **516***a* has increased in one or more of the previously allocated regular first tier containers (represented by elements **512**, **514**, **516**). If the remaining unused slack (i.e. the sum of the of the unutilized allocated resources **512***b*, **514***b*, **516***b*, and the unallocated resources **502***b*) are not sufficient to process the tasks executing in opportunistic container or the actual resource utilization has risen above a certain threshold, the scheduler **340***a* may elect to de-allocate the opportunistic container represented by element **520** as shown in FIG. **5**F.

[0068] Alternatively, if resource availability has opened up at any of the other worker nodes in the cluster, the scheduler **340***a* may opt to re-assign the requested task to another worker node and allocate a regular first tier container or opportunistic second tier container at that node to process the task. In many cases, de-allocation of an opportunistic container and the resulting termination of the task executing in the container will necessitate restarting the task at a later time (either at another worker node or in a newly allocated container in the same worker node). However, in some cases, tasks may be able to preserve any partially performed work, for example, through the use of checkpointing. Enabling the resumption of tasks after resource de-allocation coupled with cross-node container promotion would further incentivize tasks not to opt out of opportunistic scheduling.

[0069] In some embodiments preemption of a task executing in a task may not necessarily lead to de-allocation of the opportunistic container. For example, in an embodiment, the node manager **310***b* may suspend a task executing in an opportunistic container for a predetermined period of time or until actual resource utilization meets a certain condition (e.g. falls below the first threshold) so that tasks executing in the regular first tier containers are not interrupted. Once the other tasks complete or actual resources utilization conditions improve, the node manager **310***b* may again resume execution of the task previously executing the opportunistic container.

[0070] In some embodiments, the determination of whether to preempt a task executing in an opportunistic container is made by the scheduler **340***a* at the master node. For example, in an embodiment the scheduler **340***a* may de-allocate a previously-allocated opportunistic second tier container in response to determining that actual computing resource utilization at a worker node has risen above a threshold (e.g. $T_{preempt} \times R_n$) such that other tasks are not interrupted. Alternatively, in some embodiments a worker node may make this determination without any input form the master node. For example in an embodiment, a node manager **310***b* at a worker node may de-allocate a previously-allocated opportunistic second tier container in response to determining that actual computing resource utilization has risen above a threshold (e.g. $T_{preempt} \times R_n$) such that other tasks at the node are not interrupted. In such embodiments, the node manager **310***b* may inform the scheduler **340***a* that the opportunistic container was de-allocated without completing the requested task, thereby prompting the scheduler **340***a* to restart the allocation process perform the task.

[0071] In some embodiments, the variable preemption parameter $T_{preempt}$ may be set at a particular value (e.g. between 0 and 1). The $T_{preempt}$ parameter can also be generalized to a vector, with dimensions representing different resource types. While the parameter $T_{preempt}$ may be cluster-wide, in some embodiments it may be specific to a worker node given the characteristics and/or capabilities of the worker node. For example, in practical implementations, worker nodes may have different hardware, storage hotspots, and other deployed software. Further, in some embodiments, the parameter $T_{preempt}$ may be job specific. Some job types may be more amendable to opportunistic scheduling than others.

[0072] In some embodiments, preemption may be handled differently for different resource types depending on the effects of resource deprivation; specifically the extent to which resource deprivation may lead to performance degradation and failures. For example, contention over memory can easily lead to task failures. Contention over CPU resources, on the other hand, may causes performance degradation for small amounts of contention but typically will not lead to task failure unless contention is very high. In other words some computing resources (such as CPU, disk, and network) can be referred to as "malleable" while other computing resource (such as memory) may be considered "non-malleable." In certain embodiments, extra care is taken to ensure that overall utilization of non-malleable resource remains well below capacity. Accordingly, the previously described T_{alloc} and $T_{preempt}$ parameters may therefore be set on a per-resource basis as well as a per-node basis. For example, to protect against over-subscription of non-malleable resources, T_{alloc} and/or $T_{preempt}$ may be set at relatively low values. This will result in a lower willingness by the worker node to accept opportunistic scheduling of

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 211 of 411

containers for such resources as well as a higher willingness to preempt tasks executing in opportunistic containers if they are allocated.

[0073] Similar to T_{alloc}, the optimal values for $T_{preempt}$ will depend on worker node resources and the workload scheduled on the node. A high value for $T_{preempt}$ indicating a low willingness to preempt opportunistic containers will lead to fewer preemptions. This is sustainable if the utilization of running containers remains steady. However, major fluctuations in utilization without preemptions can lead to severe contention. In practice, optimal values for $T_{preempt}$ may be arrived at through trial-and-error. For example, a cluster administrator may run a series of predictable workloads and track the overall sensitivity to varying values for $T_{preempt}$, whether implemented cluster wide or at specific nodes. Tools can be provided (e.g. via a user interface **214** of a unified distributed computing platform **212**) that can enable a cluster administrator to fine tune values for $T_{preempt}$ (cluster wide, on per node basis, on a per job-type basis, etc.). To inform these adjustments by the cluster administrator, real-time and/or historical data regarding cluster utilization and performance can be provided as well.

[0074] Setting values for $T_{preempt}$ using trial-and-error may work in some implementations, however may be impractical in others. Accordingly, in some embodiments, values for $T_{preempt}$ (cluster wide, on per node basis, on a per job-type basis, etc.) can be set automatically using certain optimization techniques. For example, optimal value for $T_{preempt}$ may be determined through a feedback loop from past rims, for example by using gradient-descent.

Promotions and Opt-Out Provisions

[0075] Prioritizing regular containers helps with their predictability and performance. Opportunistic containers, on the other hand, are run at a lower priority and can be preempted more frequently. Many jobs do not have service level agreements (SLAs) and can accommodate opportunistic containers for anticipated gains in makespan. However, some jobs cannot tolerate these preemptions and need the scheduler **340***a* to offer regular containers at the same rate, irrespective of any additional opportunistic containers.

[0076] In certain cases, it is possible a job is allocated opportunistic containers, but the allocation of regular containers is delayed. Consider a job with three tasks, where each task runs for 2t units of time when run as a regular container. FIGS. **7**A and **7**B illustrate an example task schedule in such a scenario. As shown in FIG. **7**A, without the possibility of opportunistic containers, a new regular task is scheduled at regular containers R**1**-R**3** every t units of time resulting in the job taking a total of 4t units of time to complete. Alternatively with opportunistic containers enabled the third task can be scheduled as an opportunistic container O**1** on Node **3**, as shown in FIG. **7**B. As per the schedule, without opportunistic containers, the third task would have been allocated at a regular container R**3** on Node **1** at time 2t. During the time interval 2t to 3t, the job has fewer regular containers than the base case. In spite of this, the particular job represented in FIG. **7**B benefits from the opportunistic allocation finishing at time 3.5t, even though processing of the task in an opportunistic container O**1** may have taken slightly longer than in a regular container R**3**.

[0077] To address concerns related to opportunistic scheduling, a system implementing UBIS may attempt to promote opportunistic containers over regular containers. Same node promotions are typically straightforward and beneficial. The scheduler **340***a* at a master node and node manager **310***b* at a worker node can update their book-keeping and address any isolation settings. Cross node promotions may be more involved, however. If a particular job does not have a way to checkpoint and migrate the task in a work-preserving manner, the task must be killed on one node and restarted from the beginning on another node. As a result, cross-node preemption may lead to longer runtime than letting the opportunistic container run and perhaps suspending the task at the opportunistic container.

[0078] If a job's SLA requirements are tight and same node promotions alone are not enough to guarantee them, a system implementing UBIS can allow these jobs to opt out of opportunistic allocation altogether. Alternatively, instead of statically opting out of all opportunistic allocation beforehand, jobs may be configured to make elections at specific points during execution. For example, a MapReduce job with large shuffle data might not want to run reduce tasks in opportunistic containers, since the cost of preemption is high. This is because all the shuffle data for the task must be transferred again. However, opportunistic containers may be particularly well-suited to speculative tasks.

[0079] As previously mentioned, in some cases, tasks may be able to preserve any partially performed work, for example, through the use of check-pointing. Enabling the resumption of tasks after resource de-allocation coupled with cross-node container promotion would further incentivize tasks not to opt out of opportunistic scheduling.

Honoring Fairness

[0080] In some embodiments, the distributed computing cluster is configured to be utilized by multiple tenant users. In such embodiments the scheduler **340***a* may include one or more fairness schedulers configured to promote the fair sharing of computing resources among users of the cluster. In general, a fairness-based scheduler can be configured to honor one or more of the following characteristics:

- **[0081]** Sharing incentive: Users should have an incentive to share a distributed computing cluster. If there are n users sharing a cluster, each user should be allocated at least 1/n of all resources in the cluster. Otherwise, users might be better off running their own partition.
- **[0082]** Strategy-proof: A user should not get a larger allocation by misrepresenting her resource requirements. For example, a user should not get more resources just by asking for larger containers.
- **[0083]** Envy-free: A user should not prefer the allocation of another user. That is, one user should not be allocated more resources or higher quality resources than another.
- **[0084]** Pareto-efficiency: On a fully allocated cluster, one user cannot be assigned more resources without adversely affecting another user's allocation.

[0085] In some implementations, a fair scheduler in YARN can be configured to implement max-min fairness as well as one or more of the above characteristics. As previously mentioned, in the allocation of regular first tier containers, UBIS may build upon scheduling techniques implemented, for example in YARN. In other words, unallocated resources in a distributed computing cluster can be fairly allocated as regular first tier containers using existing scheduling techniques, for example included in YARN. In some embodiments, opportunistic second tier containers are allo-

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 212 of 411

cated only when the scheduler **340***a* cannot allocate any more regular first tier containers. The set of opportunistic containers can therefore be allocated using similar fairness techniques to the pool of unutilized resources, instead of unallocated resources, leading to fair allocation of unutilized resources among tenant clients. If a job opts out of opportunistic scheduling, it is not allocated an opportunistic container and its share of the unutilized resources can be distributed among the other competing jobs.

[0086] A system implementing UBIS therefore can be configured to implement any one or more of the above mentioned fairness characteristics and further incentivizes the sharing of computing resources in a cluster among users. In addition to unallocated resources, users can be allocated opportunistic containers when other users are not fully utilizing their regular containers.

Example Computing System

[0087] FIG. **8** shows a diagrammatic representation of a machine in the example form of a computer system within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein.

[0088] In the example of FIG. **8**, the computer system **800** includes a processor, memory, non-volatile memory, and an interface device. Various common components (e.g., cache memory) are omitted for illustrative simplicity. The computer system **800** is intended to illustrate a hardware device on which any of the components depicted in the example of FIG. **1** (and any other components described in this specification) can be implemented. The computer system **800** can be of any applicable known or convenient type. The components of the computer system **800** can be coupled together via a bus or through some other known or convenient device.

[0089] The processor may be, for example, a conventional microprocessor such as an Intel Pentium microprocessor or Motorola power PC microprocessor. One of skill in the relevant art will recognize that the terms "machine-readable (storage) medium" or "computer-readable (storage) medium" include any type of device that is accessible by the processor.

[0090] The memory is coupled to the processor by, for example, a bus. The memory can include, by way of example but not limitation, random access memory (RAM), such as dynamic RAM (DRAM) and static RAM (SRAM). The memory can be local, remote, or distributed.

[0091] The bus also couples the processor to the non-volatile memory and drive unit. The non-volatile memory is often a magnetic floppy or hard disk, a magnetic-optical disk, an optical disk, a read-only memory (ROM), such as a CD-ROM, EPROM, or EEPROM, a magnetic or optical card, or another form of storage for large amounts of data. Some of this data is often written, by a direct memory access process, into memory during execution of software in the computer **800**. The non-volatile storage can be local, remote, or distributed. The non-volatile memory is optional because systems can be created with all applicable data available in memory. A typical computer system will usually include at least a processor, memory, and a device (e.g., a bus) coupling the memory to the processor.

[0092] Software is typically stored in the non-volatile memory and/or the drive unit. Indeed, for large programs, it may not even be possible to store the entire program in the memory. Nevertheless, it should be understood that for software to run, if necessary, it is moved to a computer readable location appropriate for processing, and for illustrative purposes, that location is referred to as the memory in this paper. Even when software is moved to the memory for execution, the processor will typically make use of hardware registers to store values associated with the software, and local cache that, ideally, serves to speed up execution. As used herein, a software program is assumed to be stored at any known or convenient location (from non-volatile storage to hardware registers) when the software program is referred to as "implemented in a computer-readable medium." A processor is considered to be "configured to execute a program" when at least one value associated with the program is stored in a register readable by the processor.

[0093] The bus also couples the processor to the network interface device. The interface can include one or more of a modem or network interface. It will be appreciated that a modem or network interface can be considered to be part of the computer system. The interface can include an analog modem, isdn modem, cable modem, token ring interface, satellite transmission interface (e.g. "direct PC"), or other interfaces for coupling a computer system to other computer systems. The interface can include one or more input and/or output devices. The I/O devices can include, by way of example but not limitation, a keyboard, a mouse or other pointing device, disk drives, printers, a scanner, and other input and/or output devices, including a display device. The display device can include, by way of example but not limitation, a cathode ray tube (CRT), liquid crystal display (LCD), or some other applicable known or convenient display device. For simplicity, it is assumed that controllers of any devices not depicted in the example of FIG. **8** reside in the interface.

[0094] In operation, the computer system **800** can be controlled by operating system software that includes a file management system, such as a disk operating system. One example of operating system software with associated file management system software is the family of operating systems known as Windows® from Microsoft Corporation of Redmond, Wash., and their associated file management systems. Another example of operating system software with its associated file management system software is the Linux operating system and its associated file management system. The file management system is typically stored in the non-volatile memory and/or drive unit and causes the processor to execute the various acts required by the operating system to input and output data and to store data in the memory, including storing files on the non-volatile memory and/or drive unit.

Disclaimers

[0095] Some portions of the detailed description may be presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, trans-

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 213 of 411

ferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0096] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0097] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the methods of some embodiments. The required structure for a variety of these systems will appear from the description below. In addition, the techniques are not described with reference to any particular programming language, and various embodiments may thus be implemented using a variety of programming languages.

[0098] In alternative embodiments, the machine operates as a standalone device or may be connected (e.g., networked) to other machines. In a networked deployment, the machine may operate in the capacity of a server or a client machine in a client-server network environment, or as a peer machine in a peer-to-peer (or distributed) network environment.

[0099] The machine may be a server computer, a client computer, a personal computer (PC), a tablet PC, a laptop computer, a set-top box (STB), a personal digital assistant (PDA), a cellular telephone, an iPhone, a Blackberry, a processor, a telephone, a web appliance, a network router, switch or bridge, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine.

[0100] While the machine-readable medium or machine-readable storage medium is shown in an exemplary embodiment to be a single medium, the term "machine-readable medium" and "machine-readable storage medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term "machine-readable medium" and "machine-readable storage medium" shall also be taken to include any medium that is capable of storing, encoding or carrying a set of instructions for execution by the machine and that cause the machine to perform any one or more of the methodologies of the presently disclosed technique and innovation.

[0101] In general, the routines executed to implement the embodiments of the disclosure, may be implemented as part of an operating system or a specific application, component, program, object, module or sequence of instructions referred to as "computer programs." The computer programs typically comprise one or more instructions set at various times in various memory and storage devices in a computer, and that, when read and executed by one or more processing units or processors in a computer, cause the computer to perform operations to execute elements involving the various aspects of the disclosure.

[0102] Moreover, while embodiments have been described in the context of fully functioning computers and computer systems, those skilled in the art will appreciate that the various embodiments are capable of being distributed as a program product in a variety of forms, and that the disclosure applies equally regardless of the particular type of machine or computer-readable media used to actually effect the distribution.

[0103] Further examples of machine-readable storage media, machine-readable media, or computer-readable (storage) media include but are not limited to recordable type media such as volatile and non-volatile memory devices, floppy and other removable disks, hard disk drives, optical disks (e.g., Compact Disk Read-Only Memory (CD ROMS), Digital Versatile Disks, (DVDs), etc.), among others, and transmission type media such as digital and analog communication links.

[0104] Unless the context clearly requires otherwise, throughout the description and the claims, the words "comprise," "comprising," and the like are to be construed in an inclusive sense, as opposed to an exclusive or exhaustive sense; that is to say, in the sense of "including, but not limited to." As used herein, the terms "connected," "coupled," or any variant thereof, means any connection or coupling, either direct or indirect, between two or more elements; the coupling of connection between the elements can be physical, logical, or a combination thereof. Additionally, the words "herein," "above," "below," and words of similar import, when used in this application, shall refer to this application as a whole and not to any particular portions of this application. Where the context permits, words in the above Detailed Description using the singular or plural number may also include the plural or singular number respectively. The word "or," in reference to a list of two or more items, covers all of the following interpretations of the word: any of the items in the list, all of the items in the list, and any combination of the items in the list.

[0105] The above detailed description of embodiments of the disclosure is not intended to be exhaustive or to limit the teachings to the precise form disclosed above. While specific embodiments of, and examples for, the disclosure are described above for illustrative purposes, various equivalent modifications are possible within the scope of the disclosure, as those skilled in the relevant art will recognize. For example, while processes or blocks are presented in a given order, alternative embodiments may perform routines having steps, or employ systems having blocks, in a different order, and some processes or blocks may be deleted, moved, added, subdivided, combined, and/or modified to provide alternative or subcombinations. Each of these processes or blocks may be implemented in a variety of different ways. Also, while processes or blocks are at times shown as being performed in series, these processes or blocks may instead be performed in parallel, or may be performed at different times. Further any specific numbers noted herein are only examples: alternative implementations may employ differing values or ranges.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 214 of 411

[0106] The teachings of the disclosure provided herein can be applied to other systems, not necessarily the system described above. The elements and acts of the various embodiments described above can be combined to provide further embodiments.

[0107] Any patents and applications and other references noted above, including any that may be listed in accompanying filing papers, are incorporated herein by reference. Aspects of the disclosure can be modified, if necessary, to employ the systems, functions, and concepts of the various references described above to provide yet further embodiments of the disclosure.

[0108] These and other changes can be made to the disclosure in light of the above Detailed Description. While the above description describes certain embodiments of the disclosure, and describes the best mode contemplated, no matter how detailed the above appears in text, the teachings can be practiced in many ways. Details of the system may vary considerably in its implementation details, while still being encompassed by the subject matter disclosed herein. As noted above, particular terminology used when describing certain features or aspects of the disclosure should not be taken to imply that the terminology is being redefined herein to be restricted to any specific characteristics, features, or aspects of the disclosure with which that terminology is associated. In general, the terms used in the following claims should not be construed to limit the disclosure to the specific embodiments disclosed in the specification, unless the above Detailed Description section explicitly defines such terms. Accordingly, the actual scope of the disclosure encompasses not only the disclosed embodiments, but also all equivalent ways of practicing or implementing the disclosure under the claims.

What is claimed is:

1. A computer-implemented method for recouping underutilized computing resources that have been previously allocated as containers for processing tasks at a plurality of worker nodes in a distributed computing cluster, the method comprising:

receiving, by a master node, signals from a particular worker node of the plurality of worker nodes, the signals including information indicative of actual computing resource utilization at the particular worker node, the particular worker node including a previously allocated first tier container based on previously requested computing resources to process a first task, wherein the previously requested computing resources are deterministically allocated to the first tier container; and

allocating, by the master node, an opportunistic second tier container at the particular worker node to process a second task in response to determining that the actual computing resource utilization at the particular worker node is below a first threshold;

wherein the opportunistic second tier container includes underutilized computing resources previously allocated and guaranteed to the first tier container, and wherein the opportunistic second tier container is subject to de-allocation to guarantee the computing resources to the first tier container if the actual computing resource utilization at the particular worker node rises above a second threshold.

2. The method of claim **1**, further comprising:

de-allocating, by the master node, the opportunistic second tier container in response to determining that actual computing resource utilization at the particular worker node has risen above a second threshold such that the first task is not interrupted.

3. The method of claim **1**, wherein the opportunistic second tier container is de-allocated, by the particular worker node, in response to a determination, by the particular worker node, that the actual computing resource utilization at the particular worker node has risen above a second threshold, such that the first task is not interrupted.

4. The method of claim **1**, wherein the opportunistic second tier container is allocated at the particular worker node after determining, by the master node, that the particular worker node has insufficient computing resources to allocate a first tier container to process the second task.

5. The method of claim **1**, wherein the opportunistic second tier container is allocated at the particular worker node to process the second task after determining, by the master node, that the second task allows processing by opportunistic second tier containers.

6. The method of claim **1**, wherein a request received at the worker node to process a particular task in the distributed computing cluster includes an indication to allow or disallow processing of the particular task using an opportunistic second tier container.

7. The method of claim **1**, wherein the signals from the particular worker node received at the master node are periodic heartbeat signals.

8. The method of claim **7**, further comprising

determining, by the master node, whether to de-allocate the previously allocated opportunistic second tier container at the particular worker node each time a periodic heartbeat signal is received.

9. The method of claim **1**, wherein the signals from the particular worker node received at the master node include values for the first and/or second threshold.

10. The method of claim **1**, wherein the first threshold and/or second threshold are user-configurable via the master node.

11. The method of claim **1**, wherein the first threshold and/or second threshold dynamically adjust in response to changes in actual computing resource utilization at the particular worker node.

12. The method of claim **1**, wherein the first threshold and/or second threshold are specific to the particular worker node and are different than thresholds at one or more of the other plurality of worker nodes in the distributed computing cluster.

13. The method of claim **1**, wherein the second threshold is higher than the first threshold.

14. The method of claim **1**, wherein the first and second threshold are described by the following equations:

$$T_1 = T_{alloc} \times R_n$$

$$T_2 = T_{preempt} \times R_n$$

wherein T_1 is the first threshold, T_{alloc} is a variable over-allocation parameter between 0 and 1, T_2 is the second threshold, $T_{preempt}$ is a variable preemption parameter between 0 and 1, and R_n is the computing resource capacity at the particular worker node.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 215 of 411

15. The method of claim **1**, further comprising:
allocating, by the master node, an opportunistic third tier container at the particular worker node to process a third task in response to determining that the actual computing resource utilization at the particular worker node is below a fourth threshold;
wherein the opportunistic third tier container includes underutilized computing resources previously allocated and guaranteed to the first tier container and/or opportunistic second tier container, and wherein the opportunistic third tier container is subject to de-allocation to guarantee the computing resources to the first tier container and/or opportunistic second tier container if the actual computing resource utilization at the particular worker node rises above a fifth threshold.

16. The method of claim **15**, wherein the fourth threshold is the same as the first threshold, and wherein the fifth threshold is the same as the second threshold.

17. The method of claim **1**, wherein the distributed computing cluster is implemented at least in part with Hadoop and includes any one or more of:
a Hadoop Distributed File System (HDFS);
an HBase data store; or
a YARN resource manager.

18. The method of claim **1**, wherein the computing resources at the particular worker node include any one or more of processing, memory, data storage, I/O, or network resources.

19. A system for recouping underutilized computing resources that have been previously allocated as containers for processing tasks at a plurality of worker nodes in a distributed computing cluster, the system comprising:
a worker node including:
computing resources for processing the tasks, the computing resources allocatable into any one or more:
a regular first tier container; or
an opportunistic second tier container that is subject to de-allocation based on actual utilization of the computing resources to guarantee the computing resources to the regular first tier container;
a node manager configured to monitor actual utilization of the computing resources and transmit periodic heartbeat signals indicative of the actual utilization to a central resource manager for the distributed computing cluster;
a master node including the central resource manager, the central resource manager configured to:
receive a request to process a particular task in the distributed computing cluster;
allocate an opportunistic second tier container at the worker node to process the particular task in response to determining, based on signals received from the node manager, that:
the worker node has insufficient unallocated computing resources to allocate a new regular first tier container to process the task; and
the actual computing resource utilization at the worker node is below a first threshold;
wherein the opportunistic second tier container includes computing resources previously allocated to a regular first tier container to process another task; and
de-allocate the opportunistic second tier container in response to determining, based on signals received from the node manager, that actual computing resource utilization at the worker node has risen above a second threshold.

20. The system of claim **19**, wherein the distributed computing cluster is implemented at least in part with Hadoop and includes any one or more of:
a Hadoop Distributed File System (HDFS);
an HBase data store; or
a YARN resource manager.

21. The system of claim **19**, wherein the computing resources at the worker node include any one or more of processing, memory, data storage, I/O, or network resources.

22. The system of claim **19**, wherein the resource manager is further configured to determine that the particular task allows processing by opportunistic second tier containers before allocating the opportunistic second tier container to process the particular task.

23. The system of claim **19**, wherein the first threshold and/or second threshold are user-configurable via the master node and/or the worker node.

24. The system of claim **19**, wherein the first threshold and/or second threshold dynamically adjust in response to changes in actual computing resource utilization at the worker node.

25. The system of claim **19**, wherein the first threshold and/or second threshold are specific to the worker node and are different than thresholds at one or more other worker nodes in the distributed computing cluster.

27. A non-transitory computer-readable medium containing instructions for causing a central resource manager of a distributed computing cluster to:
receive signals from a particular worker node of a plurality of worker nodes in the distributed computing cluster, the signals including information indicative of actual computing resource utilization at the particular worker node, the particular worker node including a previously allocated first tier container based on previously requested computing resources to process a first task, wherein the previously requested computing resources are deterministically allocated to the first tier container; and
allocate an opportunistic second tier container at the particular worker node to process a second task in response to determining that the actual computing resource utilization at the particular worker node is below a first threshold;
wherein the opportunistic second tier container includes underutilized computing resources previously allocated and guaranteed to the first tier container, and wherein the opportunistic second tier container is subject to de-allocation to guarantee the computing resources to the first tier container; and
de-allocate the opportunistic second tier container in response to determining that actual computing resource utilization at the particular worker node has risen above a second threshold such that the first task is not interrupted.

28. The non-transitory computer-readable medium of claim **27**, wherein the distributed computing cluster is implemented at least in part with Hadoop and includes any one or more of:
a Hadoop Distributed File System (HDFS);
an H Base data store; or
a YARN resource manager.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 216 of 411

29. The non-transitory computer-readable medium of claim **27**, wherein the computing resources at the worker node include any one or more of processing, memory, data storage, I/O, or network resources.

30. The non-transitory computer-readable medium of claim **27** containing further instructions for causing the central resource manager of the distributed computing cluster to further:

determine that the second task allows processing by opportunistic second tier containers before allocating the opportunistic second tier container to process the second task.

31. The non-transitory computer-readable medium of claim **27**, wherein the first threshold and/or second threshold are user-configurable.

32. The non-transitory computer-readable medium of claim **27**, wherein the first threshold and/or second threshold dynamically adjust in response to changes in actual computing resource utilization at the worker node.

33. The non-transitory computer-readable medium of claim **27**, wherein the first threshold and/or second threshold are specific to the particular worker node and are different than thresholds at one or more of the other plurality of worker nodes in the distributed computing cluster.

* * * * *

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 217 of 411

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

MICROSOFT CORPORATION,
Petitioner,

v.

PARTEC CLUSTER COMPETENCE CENTER GMBH,
Patent Owner.

IPR2025-00318
U.S. Patent No. 11,537,442
Issued: December 27, 2022
Application No. 16/963,749
Filed: January 23, 2019

Title: Application Runtime Determined Dynamical Allocation Of Heterogeneous Compute Resources

---

PETITION FOR
***INTER PARTES* REVIEW OF U.S. PATENT NO. 11,537,442**

Darell Long, Ph.D.
09/17/2025
Exhibit F
H. Bautista, CSR #11600

TABLE OF CONTENTS

Page(s)

LIST OF EXHIBITS v

MANDATORY NOTICES UNDER 37 C.F.R. § 42.8 vii

1. Real Party-In-Interest vii

2. Related Matters vii

3. Lead And Back-Up Counsel, And Service Information vii

I. INTRODUCTION 1

II. GROUNDS FOR STANDING PER 37 C.F.R. § 42.104(a) 2

III. IDENTIFICATION OF CHALLENGE 2

A. Statement Of The Precise Relief Requested / Statutory Grounds 2

B. No Examiner Addressed These Unpatentability Grounds 2

C. *Fintiv* Denial Is Improper 4

IV. THE '442 PATENT 4

A. The '442 Patent's Specification 5

B. The Prosecution History 6

C. The Challenged Claims 7

V. LEVEL OF SKILL IN THE ART 8

A. Person Of Ordinary Skill In The Art 8

B. State Of The Art 9

VI. CLAIM CONSTRUCTION 10

VII. GROUND 1: CLAIMS 1-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE 11

A. Lippert (EX1004) 11

B. Budenske (EX1005) 13

C. Combining Lippert And Budenske 14

1. Claim 1 23

a) Element [1.1] 23

b) Element [1.2] 27

c) Element [1.3] 29

2. Claim 2 33

3. Claim 3 35

a) Element [3.1] 35

b) Element [3.2] 39

4. Claim 4 41

5. Claim 5 41

6. Claim 6 43

7. Claim 7 47

8. Claim 8 48

9. Claim 9 49

a) Element [9.1] 49

b) Element [9.2] 50

c) Element [9.3] 50

d) Element [9.4] 52

e) Element [9.5] 52

10. Claim 10 52

VIII. GROUND 2: CLAIMS 2-5 AND 8-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE AND KAMBATLA 53

A. Kambatla (EX1006) 53

B. Combining Lippert, Budenske, And Kambatla....56
1. Claim 2....62
2. Claim 3....65
a) Element 3.1....65
b) Element 3.2....66
3. Claim 4....67
4. Claim 5....68
5. Claim 8....70
6. Claim 9....71
a) Element [9.1]....71
b) Element [9.2]....71
c) Element [9.3]....71
d) Element [9.4]....72
e) Element [9.5]....72
7. Claim 10....72
IX. NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS....72
X. CONCLUSION....73
CERTIFICATE OF COMPLIANCE....74
CERTIFICATE OF SERVICE....75

TABLE OF AUTHORITIES

Page(s)

Cases

Fonar Corp. v. Gen. Elec. Co.,
107 F.3d 1543 (Fed. Cir. 1997) 21, 61

In re Fox,
471 F.2d 1405 (C.C.P.A. 1973) 23, 61

Keynetik, Inc. v. Samsung Elecs. Co.,
No. 2022-1127, 2023 WL 2003932 (Fed. Cir. Feb. 15, 2023) 21

Board Decisions

Microsoft Corp. v. Interdigital Patent Holdings, Inc.,
IPR2024-00305, Paper 12 (PTAB May 10, 2024) 73

Statutes

35 U.S.C. § 103 2

35 USC § 325 3

Rules

37 C.F.R. § 42.104 2

37 C.F.R. § 42.24 74

LIST OF EXHIBITS

No.	Description
1001	U.S. Patent No. 11,537,442 ("**'442 patent**")
1002	File History of U.S. Patent No. 11,537,442
1003	Declaration of Darrell Long, dated December 20, 2024 ("**Long Decl.**" or "**Long**")
1004	U.S. Patent App. Pub. No. U.S. 2013/0282787 ("**Lippert**")
1005	John R. Budenske et al., *A Method for the On-Line Use of Off-Line Derived Remappings of Iterative Automatic Target Recognition Tasks onto a Particular Class of Heterogeneous Parallel Platforms*, The Journal of Supercomputing, Vol. 12, No. 4 (Oct. 1998) ("**Budenske**")
1006	U.S. Patent App. Pub. No. U.S. 2018/0074855 ("**Kambatla**")
1007	Yu-Kwong Kwok et al., *A Semi-Static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems*, Journal of Parallel and Distributed Computing (2005) ("**Kwok**")
1008	Mitchell D. Theys et al., *Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach*, Solutions to Parallel and Distributed Computing Problems (A. Zomaya, F. Ercal & S. Olariu eds., 2001) ("**Theys**")
1009	U.S. Patent No. 3,496,551 ("**Driscoll**")
1010	Yuan-Chieh Chow and Walter H. Kohler, *Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System*, IEEE Transactions on Computers, Vol. C-28, No. 5 (May 1979) ("**Chow**")
1011	David M. Nicol and Paul F. Reynolds, Jr., *Optimal Dynamic Remapping of Data Parallel Computations*, IEEE Transactions on Computers, Vol. 39, No. 2 (Feb. 1990) ("**Nicol**")
1012	Arun C. Murthy, et al., Apache Hadoop YARN (2014) ("**Murthy**")
1013	Morris Jette and Mark Grondona, *SLURM: Simple Linux Utility for Resource Management*, ClusterWorld Conference and Expo ("**Jette**")

No.	Description
1014	Norbert Eicker et al., *The DEEP Project: An Alternative Approach to Heterogeneous Cluster-Computing in the Many-Core Era*, Concurrency and Computation: Practice and Experience (2016) ("**Eicker**")
1015	U.S. Patent App. Pub. No. U.S. 2017/0262319 ("**Newburn**")
1016	Carsten Clauss et al., *Dynamic Process Management with Allocation-internal Co-Scheduling towards Interactive Supercomputing*, Proceedings of the First Workshop on Co-Scheduling of HPC Applications (2016) ("**Clauss**")
1017	Declaration of Rachel J. Watters dated December 13, 2024 ("**Watters Decl.**")
1018	Samik Raychaudhuri, *Introduction to Monte Carlo Simulation*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Raychaudhuri**")
1019	David W. Bauer, Jr. and Mojdeh Mohtashemi, *An Application of Parallel Monte Carlo Modeling for Real-Time Disease Surveillance*, Proceedings of the 2008 Winter Simulation Conference (2008) ("**Bauer**")
1020	List of Challenged Claims
1021	U.S. Patent No. 5,881,227 ("**Brenner**")
1022	U.S. Patent No. 7,047,299 ("**Curtis**")
1023	N. Metropolis, *The Beginning of the Monte Carlo Method*, Los Alamos Science, Special Issue (1987) ("**Metropolis**")
1024	U.S. District Court, Eastern District of Texas, Calendar Events Set for 4/1/2026-4/30/2026, Judge Robert W. Schroeder, III, Presiding
1025	Time to Milestones Report, Judge Robert W. Schroeder, III, DocketNavigator, https://search.docketnavigator.com/patent/judge/15325/15?print=true (last visited Dec. 19, 2024)

MANDATORY NOTICES UNDER 37 C.F.R. § 42.8

1. Real Party-In-Interest

Microsoft Corporation is the sole real party-in-interest.

2. Related Matters

The '442 patent (EX1001) has been asserted in the following litigations:

- *ParTec AG et al. v. Microsoft Corp.*, Case No. 2:24-cv-00433 (E.D. Tex.), filed June 10, 2024.

3. Lead And Back-Up Counsel, And Service Information

Lead Counsel	Back-up Counsel
Andrew M. Mason (Reg. No. 64,034) andrew.mason@klarquist.com	Cameron D. Clawson, Reg. No. 73,509 cameron.clawson@klarquist.com Sarah Jelsema, Reg. No. 70,804 sarah.jelsema@klarquist.com Frank Morton-Park, Reg. No. 80,750 frank.morton-park@klarquist.com
KLARQUIST SPARKMAN, LLP 121 SW Salmon Street, Suite 1600 Portland, Oregon, 97204 503-595-5300 (phone) 503-595-5301 (fax)	

Petitioner consents to service via email at the above email addresses and the email address of Msft-Partec@klarquist.com.

Pursuant to 37 C.F.R. § 42.10(b), concurrently filed with this Petition is a Power of Attorney executed by Petitioner and appointing the above counsel.

Petitioner authorizes Account No. 02-4550 to be charged for any fees, including those enumerated in 37 C.F.R. § 42.15.

I. INTRODUCTION

Microsoft Corporation ("Petitioner") respectfully requests *inter partes* review ("IPR") of claims 1-10 of U.S. Patent No. 11,537,442 ("the '442 patent") (EX1001), allegedly assigned to ParTec Cluster Competence Center GmbH ("Patent Owner"). For the reasons set forth below, these claims should be found unpatentable and cancelled.

The challenged claims of the '442 patent relate to distributing sub-tasks of a computation task among processors of a heterogeneous computing system and adjusting the distribution for subsequent iterations of the computation task. *E.g.*, EX1001, Claim 1. As distributing sub-tasks among processors of a heterogeneous computing system was already known, the alleged point of novelty of the challenged claims was processing sub-tasks in "multiple computing iterations" and redistributing sub-tasks "based on information relating to the processing of the plurality of sub-tasks." EX1002, 174-75.

Twenty years before the filing date of the '442 patent, however, Budenske (EX1005) disclosed the same concept of processing an iterative application with a heterogeneous computing system and dynamically re-mapping sub-tasks between iterations based on "dynamic parameters of the application" relating to processing of the sub-tasks in a preceding iteration. EX1005, 391. These teachings by Budenske, when considered with more modern teachings on heterogeneous

computing systems as in Lippert (EX1004) and Kambatla (EX1006), render each challenged claim obvious.

II. GROUNDS FOR STANDING PER 37 C.F.R. § 42.104(a)

Petitioner certifies that the '442 patent is available for IPR and that Petitioner is not barred or estopped from requesting an IPR challenging the patent claims on the grounds identified in this petition.

III. IDENTIFICATION OF CHALLENGE

A. Statement Of The Precise Relief Requested / Statutory Grounds

Petitioner requests *inter partes* review of claims 1-10 (the "Challenged Claims") of the '442 patent, on the following statutory grounds:

	Reference(s)	Basis	Claims
Ground 1	Lippert (EX1004) and Budenske (EX1005)	35 U.S.C. § 103	1-10
Ground 2	Lippert, Budenske, and Kambatla (EX1006)	35 U.S.C. § 103	2-5, 8-10

For each ground, in Sections VII and VIII below, the petition presents evidence of unpatentability and establishes a reasonable likelihood that the Petitioner will prevail in establishing that each Challenged Claim is unpatentable.

B. No Examiner Addressed These Unpatentability Grounds

Prior art references Budenske (EX1005) and Kambatla (EX1006) were neither cited nor discussed during prosecution and were not referenced in the '442 patent.

Budenske provides express teachings on remapping iterative tasks onto heterogeneous parallel hardware platforms between iterations. Kambatla provides express teachings on cluster resource management. These teachings, which show that the challenged claims are unpatentable as obvious, were not considered by the Examiner.

Prior art reference Lippert (EX1004) was considered during prosecution of the '442 patent, with the Examiner rejecting all claims for being anticipated by Lippert. EX1002, 156-66.[1] Nothing in the record, however, indicates that the Examiner considered a combination of Lippert with Budenske, or of Lippert with Budenske and Kambatla, much less one informed by expert testimony, as relied on herein. By failing to recognize the long-known technique of re-distributing sub-tasks among resources of a heterogeneous computing system between iterations, as taught by Budenske, the Examiner materially erred in allowing the challenged claims.

Because these key teachings and combinations of the prior art were not previously considered by the Office, this Petition should not be denied under Section 325(d).

[1] EX1002 file history citations are to page numbers added by Petitioner.

C. *Fintiv* Denial Is Improper

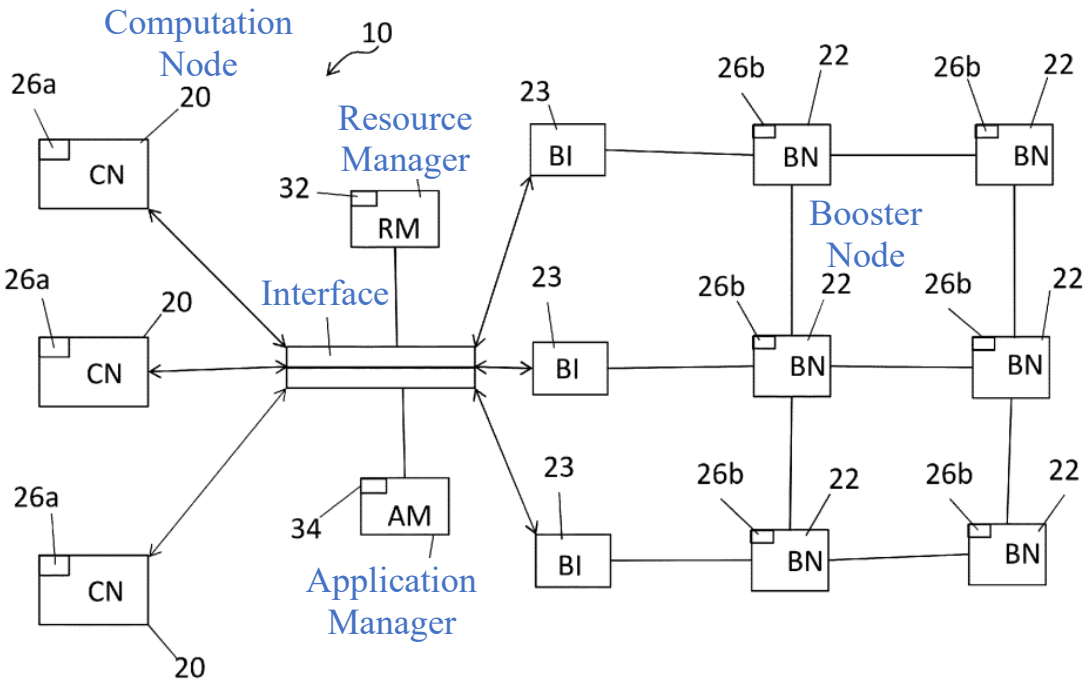
The Board should not discretionarily deny institution under *Fintiv* because Petitioner promptly filed this Petition less than four months after receiving infringement contentions on September 4, and less than two months after serving invalidity contentions on October 30. The district court case is in its early stages, before any ESI requests or production and before any depositions have been taken. While the court set a jury trial for April 20, 2026, roughly eight weeks before the expected final written decision (FWD) deadline, any actual jury trial seems likely to occur much later because (1) the district court judge has set four other cases for jury trial on April 20, 2026 (EX1024, 1-3), and (2) the district court judge's median time to jury trial is 27.0 months (EX1025, 2), which would put trial in September 2026. Finally, the Petition presents compelling merits favoring institution, as prior art Budenske was not previously considered and squarely teaches the alleged point of novelty. *Infra* Section VII.

IV. THE '442 PATENT

The '442 patent issued December 27, 2022, from a U.S. national stage application of International Patent Application No. PCT/EP2019/051615 filed January 23, 2019, and alleging priority to European Patent Application No. EP18152903.3 filed January 23, 2018. EX1001, Cover, 1:8-12.

A. The '442 Patent's Specification

The '442 patent describes a "cluster computing system 10" comprising "a number of computation nodes 20 and a number of booster nodes 22" that are "arranged to compute a computation task … comprising a plurality of sub-tasks." EX1001, 1:66-2:5, 2:66-3:2. A computation node includes, e.g., a "multi-core processor," while a booster node includes, e.g., an "accelerator-type processor." *Id.*, 3:25-27, 3:33-36. The '442 patent explains that these devices and architecture were known in the art, with booster nodes known and used for processing "highly scalable code parts" or sub-tasks of a computation task. *Id.*, 1:23-55, 3:25-38, 3:53-55.



EX1001, FIG. 1 (annotated); EX1003, ¶43.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 231 of 411

The cluster includes a resource manager and an application manager, as well as daemons associated with each element of the cluster where each daemon "runs as a background process" in the respective element. EX1001, 3:6-23. The application manager collects information from daemons at the nodes and provides this information to the resource manager "to determine if the distribution of sub-tasks between computation nodes and booster nodes can be adapted or improved upon for a further computing iteration." *Id.*, 2:13-25; EX1003, ¶43.

According to the '442 patent, "[a] job to be computed by the system may comprise a number of tasks some of which or all may be repeated a number of times during the execution of the job." EX1001, 3:44-49. The '442 patent briefly describes distributing sub-tasks between computation nodes and boosters, but that an initial distribution of sub-tasks between a computation node and a booster during a first iteration may not be optimal, so altering the distribution of the sub-tasks among nodes for later iterations "might improve the efficiency of the computation of the task." *Id.*, 3:59-4:3; EX1003, ¶44.

B. The Prosecution History

The first Office action interpreted the elements "a plurality of computation nodes, a plurality of booster nodes, communication interface, resource manager, and application manager" in presented claim 9 under Section 112(f), and rejected all presented claims 1-10 as anticipated by U.S. 2012/0317168, Driesen et al. EX1002,

123-37. The Examiner then acknowledged during an interview that the incorrect prior art reference was cited in the first Office action, and informed the Applicant that Lippert was the correct prior art reference. *Id.*, 151-52.

The second Office action maintained the interpretation of the elements of claim 9 under Section 112(f) and rejected all claims as anticipated by Lippert. EX1002, 155-66. In response, Applicant argued that the specification includes sufficient structural support for the claim limitations to avoid Section 112(f) interpretation (*id.*, 173), and that Lippert does not anticipate the claims because Lippert does not disclose processing the plurality of sub-tasks in "multiple computing iterations." *Id.*, 174 (emphasis omitted). Applicant further argued that the claims were distinguished from Lippert because they "allow for redistribution based on information relating to the processing of the plurality of sub-tasks," whereas according to Applicant, "Lippert merely proposes a load balancing based on the workload of the nodes." *Id.*, 174-75; EX1003, ¶¶45-47.

The Examiner then allowed all pending claims. EX1002, 184.

C. The Challenged Claims

Challenged claim 1 is representative and listed here, with element numbering added in the lefthand column:

[1.1]	A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being

	arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:
[1.2]	in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and
[1.3]	generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

A full claims listing is submitted as Exhibit 1020.

V. LEVEL OF SKILL IN THE ART

A. Person Of Ordinary Skill In The Art

The person of ordinary skill in the art in January 2018 ("POSITA") would have held a bachelor's degree in computer science, electrical engineering, computer engineering, or a closely related field, and approximately three years of experience working with distributed computing technologies. More education, e.g., a Master's or Ph.D., would compensate for less work experience and vice versa. EX1003, ¶¶29-31. Such education and industry experience working with distributed computing technologies would have included designing, implementing, and maintaining distributed computing systems, such as cluster computing systems, and deploying applications on such systems. The state of the art identified below reflects some of the knowledge and skill of a POSITA. EX1003, ¶¶32-42.

B. State Of The Art

POSITAs have long known how to assign tasks between multiple processors in heterogeneous computing systems. *E.g.*, EX1009, 1:70-2:4; EX1010, 354; EX1003, ¶25. A well-known "design goal of any multi-processor computing system is to achieve high overall efficiency." EX1009, 2:13-20. To achieve overall efficiency throughout processing of a task, POSITAs have long known that tasks should be assigned to processors *dynamically* because "[s]ome computations dynamically change their behavior as they progress." EX1011, 206; EX1003, ¶26.

One known type of distributed computing architecture is a "cluster," which is a collection of interconnected computing devices ("nodes") utilized as a single, unified computing resource. EX1022, 1:16-54. Clusters often use "daemons," a daemon being a background process that performs a specific task. EX1021, 7:66-67, 8:2-3, 2:54-3:18; EX1022, 3:63-4:14. As of the earliest alleged filing date, POSITAs typically used cluster resource management frameworks such as Hadoop YARN or SLURM to manage allocation of jobs to cluster resources. EX1012, xvii-xviii, 1-20, 43-58; EX1013, 3-4. Such frameworks may include a daemon at every cluster node to enable monitoring and control of the node by a central resource management daemon. EX1012, 9, 38-40; EX1013, 5-7, 20; EX1003, ¶27.

The '442 patent includes various admissions regarding the state of the art. For example, the '442 patent admits that heterogeneous computing systems comprising

a plurality of computation nodes and a plurality of booster nodes were known. EX1001, 1:23-27. The '442 patent admits that prior art reference Eicker (EX1014), describes such a heterogeneous computing system capable of "taskifying" an application into a plurality of tasks that are distributed among the nodes to enable scalability. *Id.*, 1:39-55. The '442 patent admits that Newburn (EX1015) describes "runtime processes that can fully or partially automate the distribution of data and mapping of tasks to computing resources." *Id.*, 1:56-58; *see also id.*, 5:10-24. The '442 patent further admits that resource managers for "dynamic process management" were known and "described in Clauss" (EX1016), a prior art reference which also discloses "daemons as addressed" in the '442 patent. *Id.*, 1:28-35, 3:13-17. As an example of a job with tasks that may be repeated during execution, the '442 patent suggests a "'Monte-Carlo' based simulation," a well-known method in the art that was also known for efficient processing with cluster computing systems. *Id.*, 3:44-49; EX1018, 91; EX1019, 1029-36; EX1023, 125-130; EX1003, ¶28.

VI. CLAIM CONSTRUCTION

Unless otherwise expressly discussed below, Petitioner applies the plain and ordinary meaning of all claim terms, without waiving any rights to raise additional issues of claim construction, indefiniteness, or invalidity in any litigation. EX1003, ¶48.

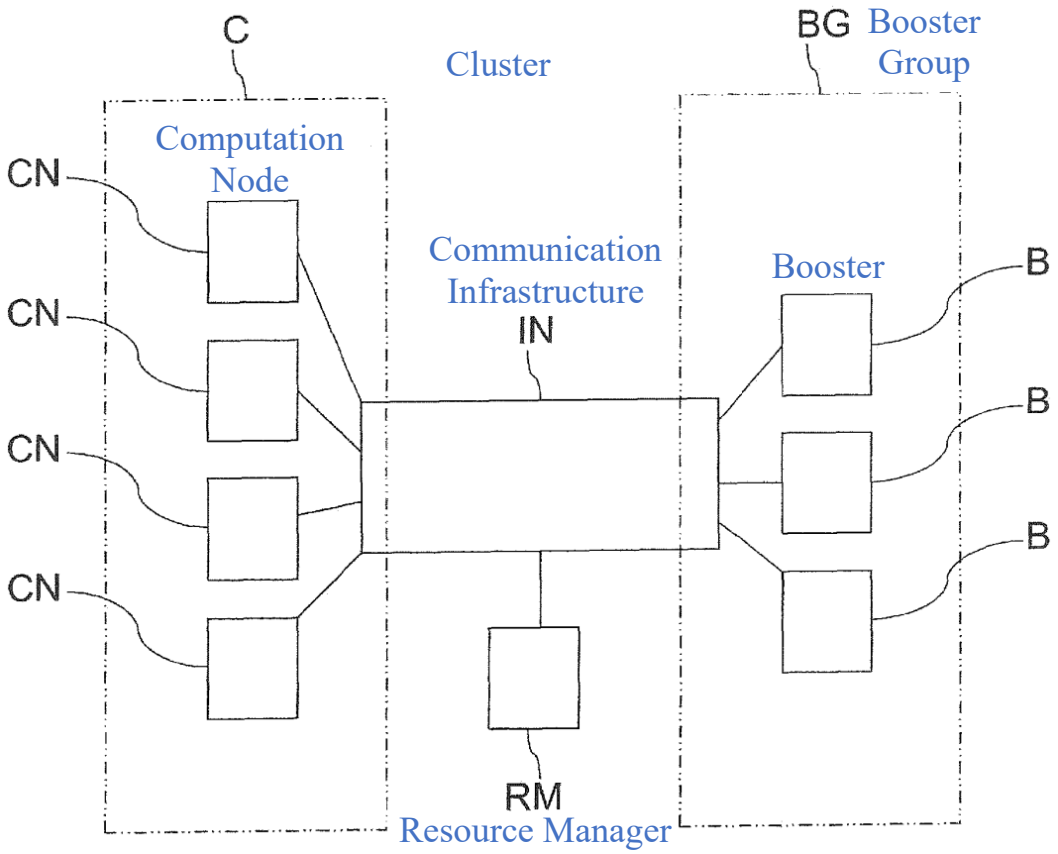
VII. GROUND 1: CLAIMS 1-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE

As explained below, claims 1-10 are obvious over Lippert in view of Budenske.

A. Lippert (EX1004)

Lippert is a U.S. patent application published on October 24, 2013, and thus qualifies as prior art under at least Section 102(a)(1). EX1004, Cover. Lippert is analogous art to the '442 patent partly because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1004, [0002]; EX1003, ¶¶49-50.

Lippert describes a "computer cluster arrangement for processing a computation task," the arrangement comprising a plurality of computation nodes and a plurality of boosters. EX1004, [0009]-[0011], FIG. 2. In Figure 2, Lippert depicts an example computer cluster with a plurality of computation nodes CN and a plurality of boosters B in a booster group BG, the computation nodes CN and the boosters B are interconnected via a communication infrastructure IN. *Id.*, [0067]. To process a computation task, a resource manager RM "initializes a first assignment and further on establishes a dynamic assignment of boosters B to computation nodes CN." *Id.*, [0068], [0013].



EX1004, FIG. 2 (annotated); EX1003, ¶51.

The computation task is allocated among computation nodes and boosters in part based on the processing requirements of each part of the computation task and the processing capacities of processing resources, where computation nodes compute a first part of the computation task while a booster computes a second part of the computation task. EX1004, [0017], [0010]-[0012], [0014]-[0016], [0018]-[0021], [0027]-[0029]; EX1003, ¶52.

B. Budenske (EX1005)

Budenske is a printed publication published October 1998 in Volume 12, Number 4 of the Journal of Supercomputing. EX1005, 387. Budenske bears a 1998 copyright and an ISSN of 0920-8542, indicia of publication. *Id.*, ii-iv. Budenske further indicates that the Journal of Supercomputing was published by Kluwer Academic Publishers, a well-known and reputable publisher, and abstracted and/or indexed in a variety of citation databases. EX1005, iii-iv. Budenske has been sufficiently accessible to the public interested in the art since its publication, and thus qualifies as prior art under Section 102(a)(1). EX1017, 1-3 (librarian attesting that Budenske was available to the public in 1998). Budenske is analogous art to the '442 patent partly because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1005, 387; EX1003, ¶53.

Budenske teaches the execution of iterative tasks with a "heterogeneous parallel hardware platform," and the real-time re-mapping of sub-tasks to processors between iterations based on the sub-tasks' processing of input data in a preceding iteration. EX1005, 387. An "Intelligent Operating System" (IOS) "make[s] a heuristically-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data." *Id.*, 387-88. A central module of the IOS (HC Kernel or HC Kernel

Monitor) (1) establishes an initial mapping of the sub-tasks onto the processing resources of the heterogeneous computing system, (2) monitors the execution of the sub-tasks, and (3) at the end of each iteration, decides how to change the mapping of the sub-tasks onto the hardware based on information about dynamic parameters obtained from processing the sub-tasks in the preceding iteration. *Id.*, 401-02; EX1003, ¶54.

Budenske teaches determining a new mapping of sub-tasks to processing resources "in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration." EX1005, 402. Budenske teaches "reconfigur[ing] the assignment of processing resources to subtasks **to reduce execution time** of the next iteration." *Id.*, 391 (emphasis added); EX1003, ¶55.

C. Combining Lippert And Budenske

A POSITA would have been motivated, with a reasonable expectation of success, to combine Lippert's teachings on a computer cluster arrangement with Budenske's teachings on re-mapping sub-tasks between iterations of an iterative application. A skilled artisan would have combined these respective teachings into an improved design (hereinafter "Lippert-Budenske") that included benefits from each reference. EX1003, ¶56.

The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
  - A plurality of computation nodes;
  - A plurality of booster nodes; and
  - A communication infrastructure interfacing each of the computation nodes and the booster nodes;
- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02); and
  - Dynamically assign, during processing of the computation task, computation nodes and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);
- An application manager module, as taught by Budenske (*e.g.*, EX1005, 388, 402), arranged to:

- Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);
- Monitor execution of sub-tasks to collect information relating to processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
- Assign, according to a new mapping obtained using the information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02).

EX1003, ¶57.

A POSITA would have been motivated, with a reasonable expectation of success, to so combine the teachings of Lippert and Budenske for at least the following reasons. EX1003, ¶58.

First, Lippert and Budenske each discloses heterogeneous computing systems that process an application or computation task with dynamic assignments. *E.g.*, EX1004, [0015]; EX1005, 387. Lippert discloses a flexible assignment of boosters to computation nodes to facilitate scalability, where parts of a computation task are allocated between computation nodes and boosters. *E.g.*, EX1004, [0013], [0015]-

[0020]. Budenske discloses a flexible assignment of sub-tasks of an iterative application to processors of a heterogeneous computing system, so that each sub-task is assigned "where it is best suited for execution" for each iteration. EX1005, 390, 387. A POSITA would have been motivated to implement the heterogeneous computing system of Lippert with the dynamic iterative sub-task assignment method of Budenske partly because doing so would facilitate efficient distributions of sub-tasks and processing resources during execution of an iterative application, thus fully leveraging the flexibility of the heterogeneous architecture and "reduc[ing] execution time" of computing iterations. EX1005, 391; EX1003, ¶59.

Second, Budenske provides express motivation for implementing its method in a heterogeneous computing system such as Lippert, namely to "fully utilize the architectural flexibility of such a [heterogeneous parallel computing] system." EX1005, 390. A POSITA would have found that the dynamic flexibility of the heterogeneous computing system taught by Lippert through the "loose coupling" of computation nodes and boosters (EX1004, [0013], [0015] (describing "a maximum flexibility in providing hardware resources")), naturally complements the dynamic flexibility of remapping sub-tasks among processing resources to account for the actual processing of the sub-tasks by the processing resources during preceding iterations, as taught by Budenske (EX1005, 391), and that the combination of these

teachings would provide increased flexibility for optimally processing iterative applications. EX1003, ¶60.

Third, Lippert recognizes that it has long been known to outsource portions of computation tasks "of high resource requirements" to accelerators (boosters), while other portions of the computation tasks are executed by computation nodes. EX1004, [0003], [0028]-[0029] (describing how computation nodes and boosters are "typically" configured with processors suited for different tasks), [0068], [0084]-[0086]. Budenske similarly recognizes the varying requirements of different computations of a given computation task, and that "[h]eterogeneous parallel architectures are ideal computing platforms for efficiently handling computational tasks with such diverse requirements." EX1005, 390 (discussing a computation task that involves "numeric computation, quasi-symbolic computation …, and symbolic computation"). A POSITA would have recognized that Lippert and Budenske follow the same principles of efficient processing of a task, comprising a plurality of sub-tasks, with a heterogeneous computing system, by assigning sub-tasks to processors (e.g., computation nodes or boosters) best suited to execute the sub-task. A POSITA thus would have been motivated to implement Budenske's teachings with a heterogeneous computing system including boosters, as taught by Lippert, partly because "the boosters are implemented to process specific problems at high speed." EX1004, [0028]; EX1003, ¶61.

Fourth, a POSITA reading Lippert would have understood the need to account for sub-task assignment among nodes for iterative applications, which is taught by Budenske. While Lippert focuses on the configuration of the processing resources of the system itself, Lippert only broadly and generally discusses the nature of the "application" or "computation task" being computed. *E.g.*, EX1004, [0015]-[0016]. Further, Lippert discloses dynamically assigning boosters to computation nodes during runtime, and outsourcing parts (i.e., sub-tasks) of the computation tasks to the assigned boosters, suggesting that sub-tasks are also dynamically remapped to newly assigned boosters during processing of the task and "in dependency of processing a specific computation task." *Id.*, [0013] ("[The resource manager may] establish a dynamic assignment at runtime, which means during processing of the computation task"), [0015] ("[B]ooster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"). Budenske, meanwhile, focuses on applications where sub-tasks are iteratively and concurrently executed, and where the processing requirements for a sub-task may evolve over time. EX1005, 387, 390. To that end, Budenske teaches monitoring dynamic parameters of the application during each iteration—where these dynamic parameters relate to the processing of the input data and "have the most impact on the execution time of the subtasks in the application"—and adjusting the assignment of sub-tasks for the next iteration accordingly based on the dynamic parameters. *Id.*,

391, 393. A POSITA thus would have naturally turned to Budenske when implementing Lippert for iterative applications, in order to "assign each … subtask to the processors where it is best suited for execution," and thus minimize execution time as the processing requirements of the input data and/or computation task evolves over time. *Id.*, 390; EX1003, ¶62.

Fifth, a POSITA would have recognized the compatibility of Lippert and Budenske's teachings. Lippert, for example, recognizes that a computation task can be divided into at least a first part for computing by computation nodes, and a second part for computation by boosters. EX1004, [0009]-[0014], [0016]-[0020], [0068]. Further, code blocks (e.g., sub-tasks) are classified as scalable or complex. *Id.*, [0084]. A POSITA would have understood that the "first part of the computation task" computed by the computation nodes includes the less scalable, complex sub-tasks, while the "second part of the computation task" computed by the booster(s) includes the highly scalable sub-tasks. *Id.*; EX1014, 5. Further, Lippert recognizes that "[e]ach code has highly scalable and less scalable complex elements," and so a POSITA would have been motivated to change which sub-tasks are assigned to a first part (computed by computation nodes) and a second part (computed by boosters), regardless of a sub-task's initial classification as complex or scalable (and thus initial mapping to computation nodes or boosters), if information relating to the

processing of the sub-task in a preceding iteration suggests that the sub-task would be better processed by a different node. EX1003, ¶63.

Sixth, nothing in these references teaches away from this combination. On the contrary, a POSITA would have recognized that the heterogeneous computing systems of Budenske and Lippert are similar. Specifically, Budenske discloses a heterogeneous computing system including DSP (digital signal processors) and RISC (reduced instruction set computer) processors (EX1005, 391), and a POSITA would have recognized that the DSPs are processors that function as accelerators/boosters and the RISC processors as computing nodes, much like the processing elements in Lippert. A POSITA thus would have understood that the teachings of Lippert and Budenske are compatible and would work well together. EX1003, ¶64.

Finally, a POSITA would have reasonably expected to succeed in implementing the Lippert-Budenske combination because doing so would only entail providing software code, which was well within the skill of a POSITA. *E.g.*, *Keynetik, Inc. v. Samsung Elecs. Co.*, No. 2022-1127, 2023 WL 2003932, at *2 (Fed. Cir. Feb. 15, 2023) (citing *Fonar Corp. v. Gen. Elec. Co.*, 107 F.3d 1543, 1549 (Fed. Cir. 1997)) ("Normally, once the function to be performed by software has been identified, writing code to achieve that function is within the skill of the art."). Further, a POSITA would have reasonably expected success implementing the

teachings of Budenske in a heterogeneous computing system as taught by Lippert partly because there is nothing incompatible between the two approaches, and because Budenske explains that its teachings "can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the application and platforms considered here." EX1005, 404. Other prior art references available to the POSITA disclosed successful implementations of Budenske's teachings and confirm that Budenske's approach to re-mapping sub-tasks is feasible and achieves faster execution times compared to other dynamic task scheduling techniques. EX1007, 78-79 (describing Budenske), 92 (illustrating execution times of Budenske's "On-Off" approach compared to other dynamic scheduling approaches for task graphs of various sizes and types); EX1008, 156-65 (discussing Budenske's approach and performance results). In view of these references, a POSITA would have reasonably expected to succeed implementing Lippert-Budenske and achieving the benefits described in each reference. EX1003, ¶65.

Further, the '442 patent provides only a high-level description of adjusting the distribution of sub-tasks between computation nodes and boosters. *E.g.*, EX1001, 3:59-4:3, 4:23-31. This level of disclosure reflects the state of the art, as "it assumes anyone desiring to carry out the process would know of the equipment and techniques to be used, none being specifically described," *In re Fox*, 471 F.2d 1405,

1407 (C.C.P.A. 1973), and thus confirms that POSITAs would have reasonably expected to succeed implementing Lippert-Budenske. EX1003, ¶66.

So combined, the Lippert-Budenske combination dynamically distributes sub-tasks of a computation task among computation nodes and boosters of a heterogeneous computation system between iterations of the computation task, in order to most efficiently process the computation task, and thus renders all challenged claims obvious. EX1003, ¶67.

1. Claim 1

a) Element [1.1] [2]

[1.1] A method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks, the method comprising:

Lippert-Budenske discloses and teaches this element[3]. Lippert and Budenske each discloses a method of operating a heterogeneous computing system to compute

[2] Reference numbers in the format of [claim#.element#] are added throughout for ease of reference.

[3] Petitioner uses the terms "limitation" and "element" loosely throughout the Petition and, unless otherwise noted, without taking positions on whether the claim language is entitled to patentable weight or is merely part of the claim's environment.

a computation task. Lippert discloses operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, with at least one of the nodes being arranged to compute a computation task comprising a plurality of sub-tasks. Specifically, Lippert discloses a "computer cluster arrangement for processing a computation task," where the "computer cluster arrangement" comprises a plurality of computation nodes and a plurality of boosters (booster nodes). EX1004, [0009]-[0013]. Processors applied in computation nodes are different from the processors applied in boosters. *Id*., [0029]. A POSITA thus would have understood that the computer cluster arrangement is a heterogeneous computing system because the computer cluster arrangement comprises different types of processors, arranged as computation nodes and booster nodes. Figure 2 of Lippert, reproduced below, "shows a computer cluster arrangement comprising a cluster C as well as a booster group BG," where the cluster as depicted comprises "four computation nodes, also referred to as CN, as well as three boosters, also referred to as B." *Id.*, [0067].

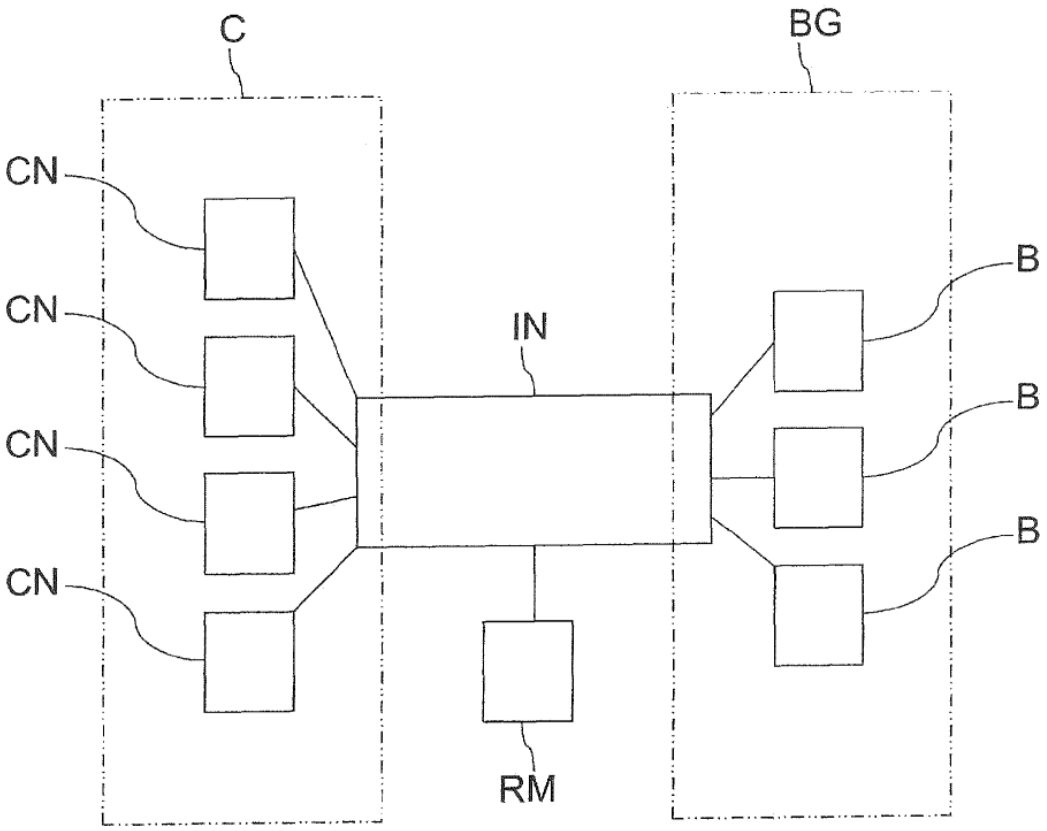


Fig.: 2

EX1004, FIG. 2; EX1003, ¶68.

Lippert discloses that "the computation task may comprise several sub problems, also referred to as sub tasks, which in their entirety describe the overall computation task." EX1004, [0016]. The computation task may be divided "into several parts," and the computer cluster arrangement is arranged "to solve the parts of the computation task in parallel or in succession." *Id.*, [0016]; EX1003, ¶69. Thus, Lippert (and Lippert-Budenske) discloses this claim element.

Budenske discloses a method for operating a heterogeneous computing system to process a computation task, the computation task comprising a plurality of sub-tasks. Budenske discloses a "methodology for real-time on-line input-data dependent remapping of the application subtasks to the processors in the heterogeneous parallel hardware platform." EX1005, 387. Budenske explains that the configuration of the heterogeneous parallel hardware platform (heterogeneous computing system) depends on the expected needs of the application (task) to be computed, and as an example teaches a heterogeneous parallel hardware platform that includes "up to four different types of processors, and up to a total of 64 processors (of all types combined)." *Id.*, 391; EX1003, ¶70.

As explained by Budenske, an "application … can be modeled as an iterative execution of a set of partially ordered subtasks." EX1005, 390. An "application task" is represented as "a *DDG* (data dependency graph), whose nodes are the subtasks that need to be executed to perform the application and whose arcs are the data dependencies among subtasks." *Id.*, 391. The DDG is "structured as a directed acyclic graph (*DAG*)." *Id.*; EX1003, ¶71.

In combination, Lippert-Budeske discloses a method of operating a heterogeneous computing system comprising a plurality of computation nodes and a plurality of booster nodes, at least one of the plurality of computation nodes and a

plurality of booster nodes being arranged to compute a computation task, the computation task comprising a plurality of sub-tasks. EX1003, ¶72.

b) Element [1.2]

[1.2] in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution; and

Lippert-Budenske discloses and teaches this element. Lippert discloses computing a "first part of a computation task" and a "second part of the computation task" (*the plurality of sub-tasks*) by, respectively, "at least two of the plurality of computation nodes CN" (*at least a portion of the plurality of computation nodes*) and "at least one booster B" (*at least a portion of the plurality of booster nodes*). EX1004, [0091]. A POSITA would have understood this computing of the parts of the computation task by the at least two of the plurality of computation nodes and the at least one booster as *assigning and processing* the parts of the computation task *in a first distribution*, at least because Lippert teaches "establish[ing] a static assignment [of a booster to a computation node] at start of a processing of a computation task," where "assignment information [is provided] to the computation nodes for outsourcing parts of the computation tasks from at least one computation node to at least one booster." *Id*., [0013]-[0014]; EX1003, ¶73.

Although Lippert discloses the computation node "outsourcing" parts of the computation task to a booster where the computation node processes the first part in

a first step and the booster node processes the second part in a second step (e.g., EX1004, [0091]-[0092]), suggesting that the parts are solved in succession, Lippert discloses also that the system may "solve the parts of the computation task *in parallel* or in succession." *Id*., [0016] (emphasis added). Further, Lippert does not expressly specifically disclose iterative execution of a computation task, and thus does not expressly specifically disclose a "first computing iteration," but Lippert more generally explains that "the person skilled in the art appreciates that any of the steps can be performed iteratively, in a different order and may comprise further sub steps." *Id.*, [0093]; EX1003, ¶74.

Budenske discloses assigning a plurality of sub-tasks to processing resources of a heterogeneous computing system for processing in a first iteration. For example, Budenske discloses, "[f]or the initial iteration through the set of subtasks" (*in a first computing iteration*), "assign[ing] resources (e.g., processors) to the subtasks." EX1005, 391; *see also id.*, 396-97 (describing how to assign sub-tasks to processors). This "initial mapping" (*first distribution*) of the subtasks (*the plurality of sub-tasks*) to the processing resources is used for a first computing iteration of the application (*computation task*). *Id.*, 395; EX1003, ¶75.

A POSITA would have implemented Lippert-Budenske to, in a first computing iteration of an iterative application, assign the plurality of sub-tasks to at least a portion of the computation nodes and at least a portion of the boosters

according to an initial mapping, where the initial mapping maps sub-tasks to processors (e.g., computation nodes or boosters), and where the plurality of sub-tasks are thus processed by their assigned computation nodes and boosters. The POSITA would have been motivated to implement Lippert-Budenske in this way at least because doing so would ensure that sub-tasks are assigned to processing resources "best suited" to compute the algorithms of the sub-tasks while accounting for the initial architecture of computation nodes and boosters, and characteristics of the application itself (e.g., what algorithms will be used to compute each sub-task; what are the data dependencies between sub-tasks; etc.). *E.g.*, EX1005, 392, 396-97. Lippert-Budenske thus teaches and renders obvious, in a first computing iteration, assigning and processing the plurality of sub-tasks by at least a portion of the plurality of computation nodes and at least a portion of the plurality of booster nodes in a first distribution. EX1003, ¶76.

c) Element [1.3]

[1.3] generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

Lippert-Budenske discloses and teaches this element. After processing of the sub-tasks with the "initial mapping," Budenske teaches "monitor[ing] the run-time values of the dynamic parameters at the end of each iteration through the underlying

DDG to make a heuristically-based decision whether to continue with the current mapping, or to select and instantiate a new mapping (for the next iteration)." EX1005, 395. Budenske thus discloses using values of dynamic parameters (*information relating to the processing of the sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes*) to generate a "new assignment" (*further distribution*) of sub-tasks to resources "to use for the next execution iteration through the subtasks" (*in a further computing iteration*). *Id.*, 391; EX1003, ¶77.

Specifically, "[a]fter each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image stream being processed." EX1005, 391. The dynamic parameters are "computed by the application as it executes." *Id.*, 393. The values for the dynamic parameters are thus obtained after processing the sub-tasks in an iteration: "After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel." *Id.*, 391. These values are an example of the claim's "information relating to the processing of the plurality of sub-tasks" for at least two reasons. First, they affect the nature of the sub-tasks being processed. For example, as dynamic parameters change, the computations of the sub-tasks change. *E.g.*, *id.*,

393 (explaining that dynamic parameters "change during run time" and "have the most impact on the execution time of the subtasks in the application"). The dynamic parameters therefore relate to the processing of the sub-tasks. Second, they are determined by the processing of the sub-tasks. *Id.* (explaining that dynamic parameters are "computed by the application as it executes"). Therefore, the result of processing the sub-tasks—the values of the dynamic parameters—is information relating to the processing of the sub-tasks. The "most recent values of such dynamic parameters" are then used to determine whether "to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration," and "[i]f it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks." *Id.*, 391. In particular, Budenske teaches generating a "vector of *D* representative values" as an "approximation of the set of current actual dynamic parameters," and using that vector to identify the optimal mapping of sub-tasks from the possible mappings (tracked in a "Mapped DDG" or "MDDG Table"). *Id.*, 402. Budenske thus teaches selecting a new mapping from a set of candidate mappings that best matches the values of the dynamic parameters (*the information*), and instantiating (*generating*) the further distribution according to the selected mapping for the next computing iteration. *Id.*, 402-03. Budenske also discloses an alternative wherein a fully "on-line heuristic" is used, such that the new

mapping is generated in real-time between iterations based on the information. *Id.*, 389-90, 398, 403; EX1003, ¶78.

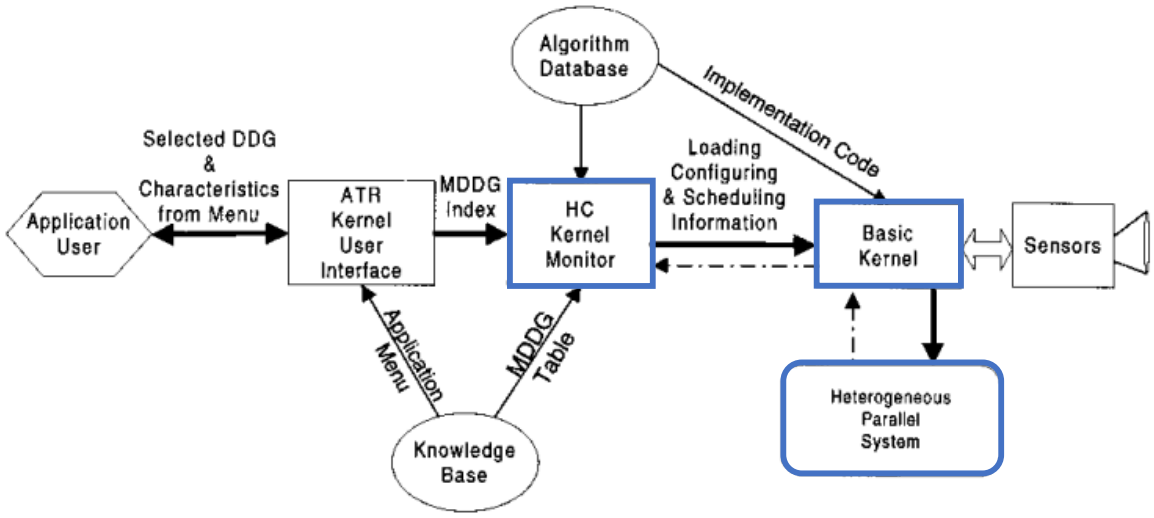
A POSITA would have found it obvious to implement the heterogeneous computing system of Lippert with the dynamic sub-task mapping method of Budenske such that, between iterations, Lippert-Budenske uses values of dynamic parameters calculated during a preceding iteration to generate a new assignment of sub-tasks to the plurality of computation nodes and the plurality of boosters for processing in the next iteration. The POSITA would have been motivated to implement Lippert-Budenske in this way partly in order "to reduce execution time of the next iteration." EX1005, 391. The POSITA would have further been motivated to implement Lippert-Budenske in this way because Budenske explains that, "[b]ecause the execution time of application subtasks … is highly input-data dependent …, this matching and scheduling of application subtasks to processors *must* be performed dynamically at run time." *Id.*, 390 (emphasis added). The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske partly because Lippert and Budenske each provide ample implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. EX1003, ¶79. Lippert-Budenske thus teaches and renders this element and this claimed method obvious.

2. Claim 2

2. The method according to claim 1, wherein an application manager receives the information and determines the further distribution.

Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske renders claim 1 obvious, as discussed *supra* Section VII.C.1. Budenske discloses an "HC Kernel" (also referred to as "HC Kernel Monitor") and a "Basic Kernel" that cooperate to manage an application, and thus a POSITA would have understood aspects of the HC Kernel and the Basic Kernel together as an *application manager*. EX1005, 388-89, 401-02. According to Budenske, "the HC Kernel interacts with the Basic Kernel … to execute the application and monitor its execution so it can decide when to remap the subtasks onto the hardware platform" (*id.*, 388), where the Basic Kernel receives "the latest values of … dynamic parameters" at the end of each iteration (*receives the information*). *Id.*, 402 (describing how the Basic Kernel "begin[s] execution of the application task"). The Basic Kernel sends the latest values of the dynamic parameters to an HC Kernel Monitor, which then "use[s] the most recent values of these dynamic parameters to estimate if changing the matching and scheduling will reduce the expected execution time of the next iteration through the corresponding DDG." *Id.* Figure 3 of Budenske depicts the Basic Kernel and the HC Kernel Monitor (together, *the application manager*) interacting with the

heterogeneous parallel computing system during execution of an iterative application:



Id., FIG. 3 (annotated); *see also id.*, FIG. 1; EX1003, ¶80.

A POSITA would have understood the HC Kernel and the Basic Kernel of Budenske together as an "application manager" that "receives the information and determines the further distribution" because the Basic Kernel receives the values of the dynamic parameters (*receives the information*) from the heterogeneous parallel computing system and inputs the dynamic parameter values to the HC Kernel to determine a new assignment of sub-tasks to processing resources (*determines the further distribution*). EX1005, 391, 402. Lippert-Budenske thus discloses and teaches this claim element and renders this claim obvious. EX1003, ¶81.

3. Claim 3

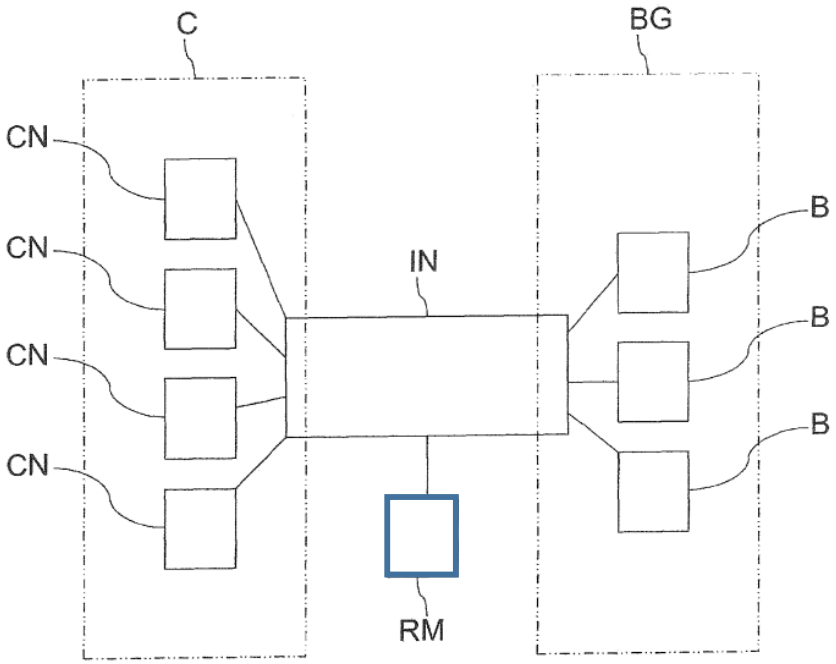
a) Element [3.1]

[3.1] The method according to claim 2, wherein a resource manager determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task and

Lippert-Budenske discloses and teaches this claim element and renders this claim obvious for the foregoing and following reasons. Specifically, as discussed below, Lippert discloses a "resource manager" and Budenske discloses an HC Kernel Monitor, and a POSITA implementing Lippert-Budenske would combine aspects of Lippert's resource manager and Budenske's HC Kernel Monitor into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task. Lippert-Budenske thus includes a *resource manager* that manages resources of the heterogeneous computing system (e.g., computation nodes, booster nodes, and the assignment thereof for processing a computation task), and an *application manager* that manages applications (e.g., computation tasks) processed by the heterogeneous computation system, where the resource manager handles an initial mapping of sub-tasks because it manages the initial arrangement of the resources and does not depend on information from processing the sub-tasks, whereas the application

manager handles re-mapping of the sub-tasks to the resources because it receives information relating to the processing of the sub-tasks during an application's execution. *Supra* Section VII.C.2; EX1003, ¶82.

Lippert discloses a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes as a function of the computation task. The resource manager "establish[es] a static assignment at start of a processing of a computation task," and "establish[es] a dynamic assignment at runtime, which means during processing of the computation task." EX1004, [0013], [0014], [0017]. FIG. 2 of Lippert depicts the resource manager RM of the heterogeneous computing system.



EX1004, FIG. 2 (annotated). This resource manager RM is similar to the resource manager RM of the '442 patent. *See* EX1001, FIG. 1 (element 28). The *resource manager* of Lippert determines the computation node-booster assignment based on "computation task requirements," and thus *as a function of the computation task*. EX1004, [0017]. Specifically, "booster allocation is performed *as a function of application needs*, which means in dependency of processing a specific computation task." *Id.*, [0015] (emphasis added). The resource manager "determin[es] an initial assignment of boosters to computation nodes" based on a predetermined assignment metric, and dynamically assigns boosters to computation nodes "at runtime" as the computation task requirements evolve and the assignment metric is accordingly altered. *Id.*, [0017], [0013], [0030]. The assignment metric is managed by the resource manager and is updated "as a function of a load balancing, which detects workload of the computer cluster arrangement, especially of the boosters." *Id.*, [0017]. For the assignment metric, Lippert teaches "detect[ing] computing capacities of boosters and furthermore detect[ing] computation task requirements," so that a booster assigned to a computation node "provides the required capacities to the computation node." *Id.* A POSITA would have understood the resource manager's assignment based on "computation task requirements" as being determined *as a function of the computation task*. *Id.*, [0017]; EX1003, ¶83.

Budenske, meanwhile, teaches an HC Kernel Monitor that determines the assignment of the plurality of sub-tasks to the plurality of nodes for the first computing iteration as a function of the computation task. Specifically, Budenske teaches an HC Kernel Monitor that "establish[es] the initial mapping" of sub-tasks to processing resources for an initial iteration (*the assignment of the plurality of sub-tasks to the plurality of computation nodes … for the first computing iteration*). EX1005, 402. The initial mapping is an MDDG for a given initial scenario (e.g., set of dynamic parameter values), where the MDDG is structured as a directed acyclic graph that arranges sub-tasks according to "data dependencies among subtasks" with algorithms selected for implementing the sub-tasks (*as a function of the computation task*). *Id.*, 391, 396, 402. A POSITA would have understood the HC Kernel Monitor partly as a *resource manager* because it "decide[s] how to assign resources (e.g., processors) to the subtasks." *Id.*, 391; EX1003, ¶84.

In combination, Lippert-Budenske includes a resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes, as taught by Lippert, for the first computing iteration, as taught by Budenske. The assignment is determined as a function of the computation task, as taught by both Lippert and Budenske, because doing so would "enforce data-dependency constraints among the subtasks" (EX1005, 397) and ensure that the sub-tasks of an iterative application are assigned

"to the processors where [they are] best suited for execution" (*id.*, 390). *See also* EX1004, [0015], [0017]. EX1003, ¶85.

b) Element [3.2]

[3.2] wherein the application manager receives the information and processes the information as input to the resource manager such that the resource manager dynamically alters further distribution during the computing of the computation task.

Lippert-Budenske discloses and teaches this element, and thus renders the claimed method obvious for the foregoing and following reasons. As discussed *supra* Section VII.C.2, Lippert-Budenske includes an application manager that receives the information (*the application manager receives the information*) and determines the further distribution (*processes the information*). Lippert-Budenske further includes a resource manager that "establish[es] a dynamic assignment at runtime, which means during processing of the computation task" (*dynamically alters further distribution during the computing of the computation task*), as taught by Lippert. EX1004, [0013]; *supra* Section VII.C.3.a). EX1003, ¶86.

A POSITA implementing Lippert-Budenske would have found it obvious to input the new mapping of sub-tasks by the application manager into the resource manager such that the resource manager dynamically alters distribution of computation nodes and booster nodes during the computing of the computation task, because doing so would utilize the "maximum flexibility in providing hardware

resources" taught by Lippert. EX1004, [0015]; *see also* EX1005, 390 (teaching to "fully utilize the architectural flexibility"). That is, in addition to re-mapping the sub-tasks to processing elements of the system as taught by Budenske, Lippert-Budenske also dynamically assigns computation nodes and boosters to each other "during processing of the computation task" as taught by Lippert (EX1004, [0013]). By providing the new mapping for the next iteration to the resource manager, Lippert-Budenske would, for example, allocate additional boosters for processing a given sub-task or otherwise re-configure computation node-booster assignment in order to implement the new mapping and "reduce execution time of the next iteration" (EX1005, 391). EX1004, [0014]-[0017], [0068]. In this way, Lippert-Budenske "provide[s] fault tolerance in case of accelerator failures" and avoids "a lack of resources or … an excessive supply of resources" when executing an application. EX1004, [0003]. EX1003, ¶87.

A POSITA would have understood that "the resource manager dynamically alters further distribution" as recited in this claim element *encompasses* "the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other," because claim 4 depending from claim 3 specifies this, and further because the '442 patent describes the resource manager adjusting booster assignments according to updated assignment information from the application manager. EX1001, 4:27-31. Thus, in view of the

above, Lippert-Budenske discloses and teaches this element and this claim. EX1003, ¶88.

4. Claim 4

> 4. The method according to claim 3, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task.

Lippert-Budenske discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. As discussed *supra* Section VII.C.3, the resource manager of Lippert-Budenske receives the information and, if appropriate, dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. EX1004, [0013], [0017]; EX1003, ¶89.

5. Claim 5

> 5. The method according to claim 1, wherein daemons operate in the plurality of computation nodes and the plurality of booster nodes to generate the information.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons addressing claim 1 and for the following reasons. Lippert-Budenske renders claim 1 obvious. *Supra* Section VII.C.1. Lippert discloses "detect[ing] computing capacities of boosters" (EX1004, [0017]), and Budenske teaches values of dynamic parameters being reported to the Basic Kernel

during each iteration (*e.g.*, EX1005, 402). In order to send the dynamic parameter values taught by Budenske, as well as information regarding computing capacities as taught by Lippert, from each computation node and each booster node to a central manager to determine a re-mapping of sub-tasks, a POSITA thus would have implemented Lippert-Budenske with daemons at each computation node and each booster node configured to collect and send this information. POSITAs knew that a daemon is a background process that performs a specific task, and that daemons were commonly implemented at compute nodes to execute tasks at the compute nodes and report status information back to a central resource manager for managing cluster resources. *E.g.*, EX1013, 5-7 (describing a central controller daemon and daemons running at each compute node "to export control to SLURM"), 20 (describing a daemon at each node "for managing user jobs and monitoring system state," where its "most common action … is to report system state on request"); EX1021, 7:66-67 (defining a "daemon" as "a program that runs unattended to perform a standard service"), 8:2-3 ("Daemon processes generally provide services that must be available at all times."), 2:54-3:18 (describing using daemons to facilitate communication in a "parallel processing environment" comprising "a plurality of nodes connected through a network to one another"). A POSITA thus would have been motivated to implement background processes (i.e., daemons) operating at each node that communicate the relevant information in a status update

for dynamically remapping the sub-tasks among the nodes, to beneficially facilitate inter-element communication while providing communication authentication, and to enable independent control of task execution and reporting of task processing at each node. *E.g.*, EX1012, 9 (describing a central daemon and node daemons or "TaskTrackers" for executing tasks at the node), 38-40 (describing a "NodeManager" daemon at each node "responsible for launching the applications' containers, monitoring their resource usage …, and reporting the same to the ResourceManager"); EX1013, 10 (describing how daemons authenticate and secure communications between nodes); EX1022, 3:63-66 ("As is commonly known, a daemon is a process that runs in the background and performs a specified operation at predefined times or in response to certain events."), 3:66-4:14 (explaining that "[t]ypical daemon processes … perform administrative tasks" and are implemented in "any environment … which is capable of implementing background processes and supports communications between the various Nodes of the Cluster(s)"); EX1003, ¶90.

6. Claim 6

6. The method according to claim 1, wherein the first distribution is determined based on a rating provided in source code for each sub-task in the plurality of sub-tasks.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the above reasons associated with claim 1 and the following

reasons. Lippert discloses determining the first distribution based on a rating provided in source code for each sub-task in the plurality of sub-tasks. EX1003, ¶91.

Regarding *a rating*, Lippert discloses priority, complexity, and scalability information (individually and together, "*a rating*") for each sub-task because the sub-tasks comprise the priority information, complexity information, and scalability information. Specifically, Lippert discloses that "[s]pecific *computation tasks may comprise priority information*, which indicates how urgently this specific computation task has to be computed." EX1004, [0020] (emphasis added). This priority indicates "how urgent a processing of a computation task, or at least a part of a computation task, is compared to other parts of computation tasks being originated from other computation nodes." *Id.* This relates to the dependency between sub-tasks, where the processing of one sub-task depends on the processing of another sub-task. Regarding complexity information and scalability information, Lippert discloses applications or computation tasks where "[e]ach code has highly scalable and less scalable complex elements," broken down into "Exascale Code Blocks" and "complex Code Blocks," respectively (i.e., scalable sub-tasks and less-scalable complex sub-tasks). *Id.*, [0084]. As "the distinction between highly scalable and complex is made on the level of code blocks," the code blocks (sub-tasks) include the complexity information and scalability information. *Id.* Each sub-task of

the computation task thus comprises priority information, complexity information, and scalability information (*a rating*). EX1003, ¶92.

The rating is *provided in source code for each sub-task in the plurality of sub-tasks* because the computation task "comprises" the information and is "defined by means of … a source code." EX1004, [0016]; EX1003, ¶93.

Regarding *the first distribution is determined based on a rating*, Lippert further discloses a "predetermined assignment metric" for assigning nodes to each other, with parts of the computation task allocated accordingly, being specified as a function of "complexity information, scalability information," and "priority information" (*a rating*). EX1004, [0023], [0018], [0043]. This information is specific to each sub-task, and the assignment in Lippert, including the initial distribution of sub-tasks among the nodes, is based partly on this information. *E.g.*, *id.*, [0015] ("[B]ooster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task."), [0017]-[0020] (describing using the assignment metric to assign nodes). Lippert teaches adapting the architecture of the computation nodes and the boosters of a computing cluster to account for the scalability of the code blocks. *Id.*, [0084]-[0088]; EX1003, ¶94.

A POSITA would have understood that Lippert thus discloses and teaches determining the first distribution of sub-tasks based on a rating provided in source

code for each sub-task, including a rating relating to priority, complexity, and scalability, to ensure that the part of a computation task distributed to a booster is suitable for the resource capacities of the booster. This is consistent with the disclosure in the '442 patent of an operator estimating a scalability factor for each sub-task and distributing the sub-tasks accordingly. EX1001, 4:16-47. Further, the '442 patent confirms that POSITAs already knew how to annotate application code to indicate that a given task is "highly scalable" and thus suitable for execution by boosters. EX1001, 1:39-55 (citing EX1014); EX1014, 15-16 (prior art describing how to designate portions of code as sub-tasks as highly scalable and as suitable for execution on specified hardware devices, e.g., boosters); EX1003, ¶95.

A POSITA would have found it obvious when implementing Lippert-Budenske to determine the initial mapping of sub-tasks to computation nodes and boosters based on a rating provided in source code for each sub-task, where such rating comprises an indication of priority, complexity, and scalability, as taught by Lippert. The POSITA would have been motivated to do so to create optimal assignments of sub-tasks to processing resources (e.g., computation nodes or boosters) that account for suitability of the processing resource for the corresponding scalability/complexity/priority of the sub-task while minimizing the expected execution time. The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske in this way because Lippert provides

implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. EX1003, ¶96. Lippert-Budenske thus renders this claim obvious.

7. Claim 7

7. The method according to claim 1, wherein the information is used to provide a grouping of sub-tasks in at least one of the first computing iteration and the further computing iteration.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the foregoing reasons associated with claim 1 and the following reasons. Regarding *a grouping of sub-tasks*, Lippert discloses "divid[ing] the computation task into several parts," including a first part computed by computation node(s) and a second part computed by booster(s). EX1004, [0016], [0009]-[0012]. A POSITA would have understood these "parts" of the computation task as groupings of sub-tasks, e.g., a first grouping of sub-tasks for processing by computation nodes and a second grouping of sub-tasks for processing by boosters, because Lippert discloses that a computation task "comprise[s] several sub problems, also referred to as sub tasks." *Id.*, [0016]; EX1003, ¶97.

As discussed *supra* Section VII.C.1.c), Lippert-Budenske discloses and teaches using information relating to the processing of the plurality of sub-tasks to generate a further distribution of the sub-tasks for a further computing iteration. In view of this disclosure and teaching, a POSITA implementing Lippert-Budenske

would have been motivated, with a reasonable expectation of success, to use the values of the dynamic parameters (*the information*) from a preceding iteration to provide, via the new mapping of sub-tasks, a first grouping of sub-tasks and a second grouping of sub-tasks for processing respectively with computation node(s) and booster(s) in the next iteration (*the further computing iteration*), at least because Lippert expressly discloses providing such groupings of sub-tasks. EX1004, [0016], [0009]-[0012]; EX1003, ¶98.

8. Claim 8

8. The method according to claim 3, wherein a daemon operating at a node generates a measure of a loading of the node during processing of a sub-task of the plurality of sub-tasks.

Lippert-Budenske discloses and teaches this claim element, and renders this claim obvious for the reasons associated with claims 1-3 and 5, and the following reasons. Lippert-Budenske renders claim 3 obvious, and further renders the daemons operating at the plurality of computation nodes and booster nodes obvious. *Supra* Sections VII.C.3, VII.C.5. Lippert discloses "detect[ing] workload of the computer cluster arrangement, especially of the boosters," as well as "computing capacities." EX1004, [0017]. For Lippert's resource manager to detect such information, a POSITA would have understood that a background process (i.e., a daemon) implemented at each node to measure and report workload and computing capacity information (*generates a measure of a loading of the node during processing of a*

sub-task of the plurality of sub-tasks) would be the most common way to generate and communicate such information, and the easiest to implement because daemons already exist for performing this function. *E.g.*, EX1012, 38 (describing a per-node daemon "monitoring their resource usage (CPU, memory, disk, network) and reporting the same to the ResourceManager"); EX1013, 6-7 (describing a "daemon running on each compute node"), 19-20 (describing how node daemons provide "resource consumption information … associated with a job"); EX1003, ¶99.

A POSITA thus would have been motivated, with a reasonable expectation of success, to implement Lippert-Budenske with a daemon operating at each computation node and booster node that generates a measure of a loading (workload) of the node during processing of a sub-task of the plurality of sub-tasks, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system, as taught by Budenske, to also account for "load balancing" as taught by Lippert. EX1004, [0017]; EX1003, ¶100.

9. Claim 9

a) Element [9.1]

[9.1] A heterogeneous computing system comprising:

Lippert-Budenske discloses and teaches this element, at least because Lippert and Budenske each disclose a heterogeneous computing system. *Supra* Section VII.C.1.a). EX1003, ¶101.

b) Element [9.2]

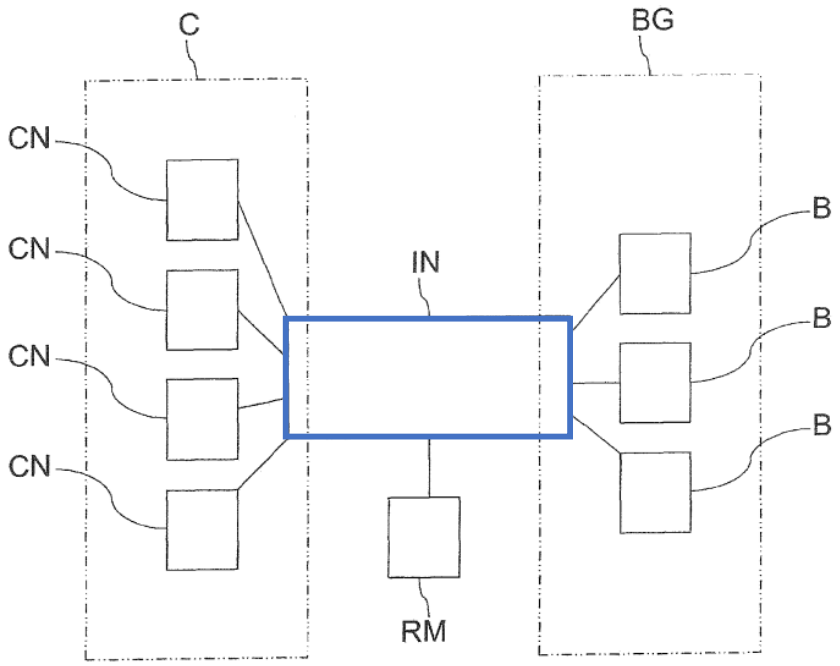
[9.2] a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks;

Lippert-Budenske discloses and teaches this element, at least because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. *Supra* Section VII.C.1.a). EX1003, ¶102.

c) Element [9.3]

[9.3] a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes;

Lippert-Budenske discloses and teaches this element, at least because Lippert discloses the plurality of computation nodes and the plurality of booster nodes "interfacing a communication infrastructure." EX1004, [0010]-[0011], [0016]. Thus, "the computation nodes as well as the boosters interact by means of the communication infrastructure." *Id.*, [0016]. FIG. 2 of Lippert depicts the communication infrastructure IN, also referred to as an interconnect, interfacing with each computation node CN and each booster B. *Id.*, [0067], FIG. 2.



EX1004, FIG. 2 (annotated); EX1003, ¶103.

Budenske similarly discloses that "[a]ll the processors of all types will have communication paths to one another," via a "bus [that] can be used to provide communications among different collections of processors." EX1005, 392. EX1003, ¶104.

Thus, when implementing Lippert-Budenske, a POSITA would have found it obvious for the heterogeneous computing system to include a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes, at least because Lippert and Budenske each disclose such a communication interface. EX1003, ¶105.

d) Element [9.4]

[9.4] a resource manager for assigning at least a portion of the plurality of booster nodes and at least a portion of the plurality of computation nodes to each other for the computing of the one or more tasks in a first computing iteration; and

Lippert-Budenske discloses and teaches this element. *Supra* Sections VII.C.3, VII.C.4. EX1003, ¶106.

e) Element [9.5]

[9.5] an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2. Lippert-Budenske thus discloses and teaches all elements and renders this claimed system obvious. EX1003, ¶107.

10. Claim 10

10. The heterogeneous computing system according to claim 9, wherein the resource manager receives the information such that the resource manager dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other.

Lippert-Budenske discloses and teaches this element, and thus renders this claim obvious. *Supra* Sections VII.C.9, VII.C.4; EX1003, ¶108.

VIII. GROUND 2: CLAIMS 2-5 AND 8-10 ARE OBVIOUS OVER LIPPERT IN VIEW OF BUDENSKE AND KAMBATLA

As explained below, claims 2-5 and 8-10 are obvious over Lippert in view of Budenske and Kambatla. While Lippert and Budenske in combination render claims 1-10 obvious, the express teachings of Kambatla regarding resource management daemons reflects the knowledge of a POSITA and further confirms the obviousness of the claims. EX1003, ¶109.

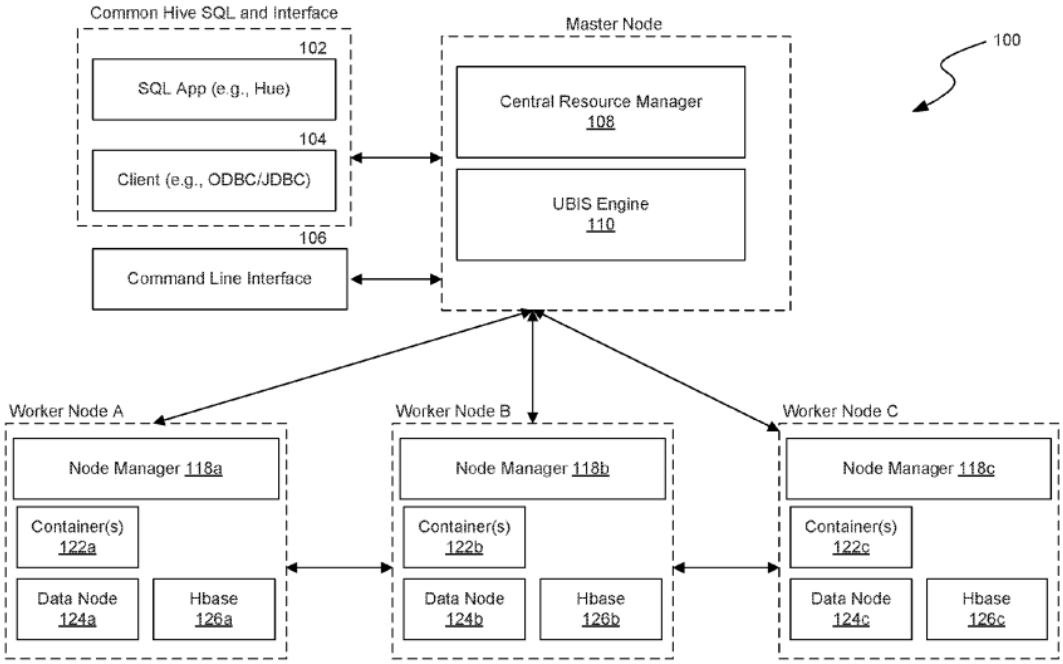
A. Kambatla (EX1006)

Kambatla is a U.S. patent application filed May 15, 2017, and claiming priority to a provisional application filed September 14, 2016, and thus qualifies as prior art under at least Section 102(a)(2). EX1006, Cover. Kambatla is analogous art to the ’442 patent because they are both in the same field of endeavor of executing computation tasks within a heterogeneous computing environment. EX1001, 1:16-20; EX1006, [0002], [0004]; EX1003, ¶110.

Kambatla discloses a resource management system for the “allocation and management of computing resources in distributed computing clusters.” EX1006, [0002]. Kambatla teaches techniques referred to as “utilization-based incremental scheduling” (UBIS) that “opportunistically allocate[s] computing resources not utilized by prior allocations.” *Id.*, [0017]. Kambatla proposes UBIS as an improvement to known “cluster scheduler” frameworks such as Apache Hadoop

YARN ("Yet Another Resource Negotiator"), used for scheduling (i.e., assigning) multiple tasks of a job to cluster computing resources. *Id.*, [0004], [0020] ("UBIS can be implemented in or for use with Apache YARN"), [0023]-[0024], [0085]. EX1003, ¶111.

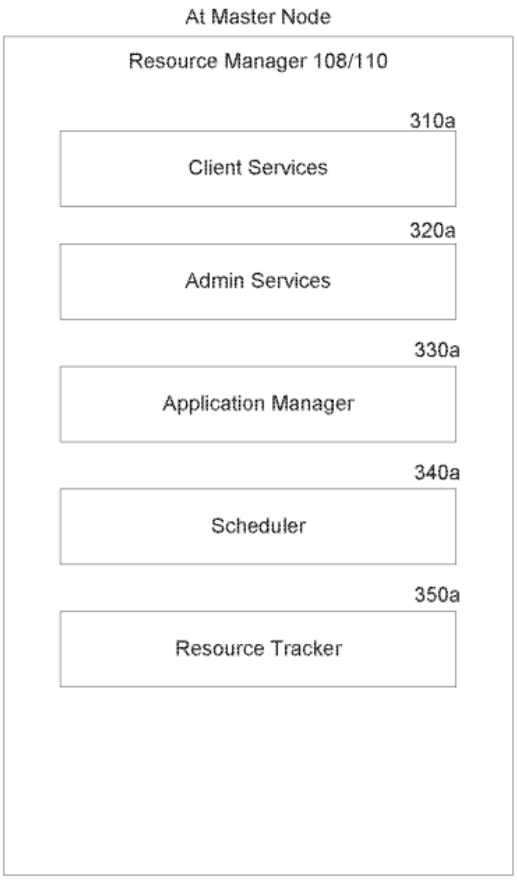
Kambatla discloses a computing environment, depicted in Figure 1, comprising a master node and a plurality of worker nodes in communication, where the master node assigns tasks of a job to the computing resources of the worker nodes for execution. EX1006, [0021]-[0022].



Id., FIG. 1. To facilitate allocation of computing resources for tasks, Kambatla discloses daemons at each computing node, including a central resource manager daemon at the master node and a node manager daemon at each worker node. *Id.*,

[0030]-[0036] (describing resource manager), [0037]-[0041] (describing node managers), FIGS. 3A-B. The resource manager is "configured to manage and arbitrate resources among applications in the system." *Id.*, [0023]. EX1003, ¶112.

Although Figure 1 depicts the resource manager 108 and the UBIS engine 110 as separate components of the master node, Figure 3A of Kambatla discloses a single resource manager comprising both components. EX1006, [0023], [0030], FIG. 3A. The resource manager comprises, *inter alia*, "an application manager 330a, a scheduler 340a, and a resource tracker 350a." *Id.*, [0031].



EX1006, FIG. 3A; EX1003, ¶113.

B. Combining Lippert, Budenske, And Kambatla

A POSITA would have found it obvious to implement a combination of Lippert and Budenske, as discussed *supra* Section VII.C. A POSITA further would have been motivated, with a reasonable expectation of success, to combine the teachings of Lippert-Budenske with Kambatla's resource scheduling daemons. A skilled artisan would have combined these respective teachings into an improved design (hereinafter "Lippert-Budenske-Kambatla") that included benefits from each reference. The resulting combination has the following teachings and elements cooperating as follows:

- A computer cluster, as taught by Lippert, including:
  - A plurality of computation nodes;
  - A plurality of booster nodes; and
  - A communication infrastructure interfacing each of the computation nodes and the booster nodes;
- A resource manager module implemented as a daemon, as taught by Lippert (*e.g.*, EX1004, [0014]), arranged to:
  - Assign booster nodes to computation nodes for processing a computation task, as taught by Lippert (*e.g.*, EX1004, [0014]);
  - Assign, according to an initial mapping, sub-tasks of the computation task to the nodes for processing in a first iteration of

the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02); and

- Dynamically assign, during processing of the computation task, computation nodes and booster nodes to each other, as taught by Lippert (*e.g.*, EX1004, [0013]);

- An application manager module, as taught by Kambatla, arranged to:
  - Deploy the sub-tasks to the computation nodes and the booster nodes, as assigned for a current iteration, for execution of the sub-tasks by the nodes in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402);
  - Monitor execution of sub-tasks to collect information relating to processing of the sub-tasks in the current iteration, as taught by Budenske (*e.g.*, EX1005, 402); and
  - Assign, according to a new mapping obtained using information related to the processing of the sub-tasks, the sub-tasks to the nodes for a subsequent iteration of the computation task, as taught by Budenske (*e.g.*, EX1005, 401-02);
- A node manager daemon at each computation node and booster node, as taught by Kambatla, arranged to:

- Generate information relating to the processing of a sub-task at the node; and
- Report the information to the application manager module.

EX1003, ¶114.

A POSITA would have been motivated, with a reasonable expectation of success, to so combine Lippert, Budenske, and Kambatla by implementing the computer cluster of Lippert with the dynamic re-mapping of sub-tasks between iterations of an iterative application as taught by Budenske, and with the resource management daemons as taught by Kambatla, for at least the following reasons. EX1003, ¶115.

First, Kambatla, like Lippert-Budenske, discloses a heterogeneous computing system that processes a task comprising a plurality of sub-tasks. EX1006, [0004], [0021], [0059]. Kambatla further teaches that such systems enable processing of not just one task or application, but multiple tasks concurrently, with the processing resources of the system being shared by the multiple tasks, and provides techniques for optimally scheduling tasks across the processing resources. *Id.*, [0004], [0017]. A POSITA thus would have been motivated to incorporate the complementary teachings of Kambatla into Lippert-Budenske in order to better leverage the full processing capacity of the heterogeneous computing system. EX1003, ¶116.

Second, a POSITA implementing Lippert-Budenske would have recognized the need to facilitate inter-element communication in a way that allows the resource manager to efficiently allocate sub-tasks to nodes, and Kambatla teaches how to do so. For example, while Lippert discloses the computation nodes, booster nodes, and resource manager communicating with each other via the communication infrastructure (*e.g.*, EX1004, [0016]), Kambatla describes a node manager daemon operating at each node to communicate with a central resource manager daemon and to manage the execution of tasks at the node. EX1006, [0023], [0038]-[0041]. A POSITA thus would have naturally turned to the teachings of Kambatla for implementation details on communicating information (i.e., values of dynamic parameters, processing capacities of the node) to the central resource manager. EX1003, ¶117.

Third, the particular teachings of Kambatla follow the same principles as Lippert-Budenske regarding the utilization of processing resources. For example, Kambatla describes the prior art background of allocating cluster computing resources to "jobs [that] typically include multiple tasks," and the problem of "resource fragmentation and severe under-utilization of the computing resources in the cluster." EX1006, [0003]-[0005]; *see also* EX1004, [0005] (discussing the problem of "over- or under subscription of accelerators Acc depending on the computation task"). This occurs because "[t]he amount of computing resources

required to process a given task can be difficult to predict," as "[i]t is inevitably difficult to accurately estimate the resource requirements of a job or its constituent tasks because: (i) resource usage of a task varies over time, and (ii) resource usage can vary across tasks of the same job based on the input they process." EX1006, [0005]. Kambatla addresses this problem by monitoring "actual utilization of resources at a computing node" via a YARN-based resource manager that communicates with node managers at nodes of a cluster, and "improv[ing] effective resource utilization in a distributed computing cluster." EX1006, [0017], [0042]; *id.*, [0018]-[0023], [0030]-[0036] (describing a central resource manager daemon), [0037]-[0041] (describing a node daemon), [0042]-[0049] (describing "Optimizing Resource Utilization"). This utilization-based resource allocation for applications deployed to a computer cluster is complementary to the dynamic assignments of Lippert-Budenske. *E.g.*, EX1004, [0015] (teaching "booster allocation is performed as a function of application needs, which means in dependency of processing a specific computation task"); EX1005, 390-91 (describing "input-data dependent" execution of sub-tasks and the remapping of sub-tasks to processors as dynamic parameters evolve over time). Thus, a POSITA seeking to fully leverage the processing capacity of a computer cluster, for example by processing multiple iterative applications concurrently, would naturally seek out and incorporate

teachings on cluster resource management daemons that facilitate dynamic resource allocation, such as those taught by Kambatla. EX1003, ¶118.

Finally, a POSITA had a reasonable expectation of success implementing Lippert-Budenske-Kambatla, at least because a POSITA was familiar with and understood how to implement "known resource management techniques (such as in YARN)" (EX1006, [0023]), and because doing so would involve writing only minor software code that was well within the skill of a POSITA. *E.g.*, *Keynetik*, 2023 WL 2003932, at *2 (citing *Fonar Corp. v. Gen. Elec. Co.*, 107 F.3d 1543, 1549 (Fed. Cir. 1997)); *see also* EX1012, xviii (explaining that "YARN makes the effort of building [distributed computing] systems significantly simpler"). Further, a POSITA would have reasonably expected to succeed incorporating Kambatla's teachings into Lippert-Budenske because Kambatla explains that its teachings can be successfully implemented in heterogeneous computing systems comprising "CPU and memory resources" (i.e., computation nodes) and "GPU[s] (graphical processing [units])" (i.e., boosters). EX1006, [0020], [0023], [0059]. Further, the '442 patent admits that the use of daemons at nodes to facilitate communication was already known, confirming that a POSITA knew how to implement such features. EX1001, 3:6-18, 3:39-43, 4:32-37; *In re Fox*, 471 F.2d at 1407; EX1003, ¶119.

Thus, a POSITA implementing the dynamic assignment of sub-tasks between iterations, as taught by Budenske, to computation nodes and boosters of a

heterogeneous computing system, as taught by Lippert, would have been motivated to implement a cluster scheduling framework as taught by Kambatla, in order "to improve effective resource utilization" and thus minimize "processing duration." EX1006, [0042]; EX1003, ¶120.

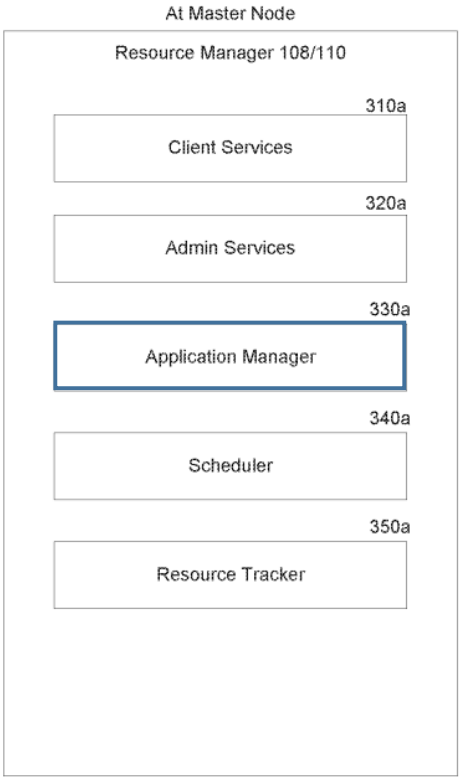
1. Claim 2[4]

Lippert-Budenske-Kambatla discloses and teaches this element and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske renders claim 1 obvious, as discussed *supra* Section VII.C.1. Further, Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2; EX1003, ¶121.

Moreover, Kambatla expressly discloses various modules of a central resource manager that a POSITA would have understood as the "application manager" recited in the claim. Kambatla discloses an "application manager 330a … responsible for maintaining a collection of submitted 'applications' or 'jobs.'" EX1006, [0034]. The application manager is a module of a "resource manager daemon." *Id.*, [0031]. The application manager includes an "application master service responsible for communicating with all the application masters" at the

[4] The full claim language is included in mapping the claims for Ground 1 (*supra* Section VII) and in the Listing of Claims (EX1020). Petitioner hereinafter does not reproduce the claim language, but instead references it by element/claim number.

computing nodes. *Id.*, [0034]. FIG. 3A of Kambatla depicts the resource manager 108 including the application manager 330a.



Id., FIG. 3A (annotated). Further, the resource tracker 350a of the resource manager "monitors available resources at the nodes, by receiving status updates from the worker nodes," while "[t]he scheduler 340a is responsible for allocating resources to the various applications." EX1006, [0035]-[0036]; *id.*, [0058] (describing how the resource manager "can actively monitor resource usage of each container at the worker nodes and of the worker nodes themselves," via "heartbeat information" that includes "utilization information"). Despite being depicted as separate components, Kambatla teaches that the resource manager can "include more or fewer

components, organized in various configurations." EX1006, [0030]. A POSITA thus would have understood that the functions of the application manager 330a, the scheduler 340a, and the resource tracker 350a were combinable into a single component (*application manager*) that manages an application deployed to cluster resources, communicates with the resources to monitor the status of the application, and assigns sub-tasks of the application to the resources for processing. EX1003, ¶122.

In combination, a POSITA implementing Lippert-Budenske-Kambatla would have found it obvious to use an application manager as taught by Kambatla to perform the functions of the HC Kernel and Basic Kernel of Budenske as discussed *supra* Section VII.C.2, at least because the application manager and the Kernels similarly manage the execution of an application on a plurality of computing devices. Further, a POSITA would have been motivated to implement the dynamic assignment of sub-tasks to the compute nodes and booster nodes of a heterogeneous computing system, as taught by Lippert-Budenske, with the application manager of Kambatla in order to efficiently enable and manage the execution of multiple such applications on the plurality of computing resources. *E.g.*, EX1006, [0004], [0023]-[0024], [0034]; EX1003, ¶123.

The POSITA would have reasonably expected to succeed because implementing such an application manager would entail only providing simple

software code and configurations to perform the known functions, which was within the skill of a POSITA. *E.g.*, EX1006, [0101] (explaining broadly how "the routines executed to implement the embodiments of the disclosure[] may be implemented"); EX1012, 59-84 (detailing how to install and configure YARN on a cluster); EX1003, ¶124.

2. Claim 3

a) Element 3.1

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. *Supra* Section VIII.B.1 (claim 2). Lippert-Budenske discloses and teaches this element, as discussed *supra* Section VII.C.3. Further, Kambatla discloses a "general resource manager configured to manage and arbitrate resources among applications in the system." EX1006, [0023], FIGS. 1, 3A. A POSITA implementing Lippert-Budenske-Kambatla would have combined the resource manager of Lippert-Budenske with the central resource manager of Kambatla into a combined resource manager that determines the assignment of the plurality of sub-tasks to the plurality of computation nodes and the plurality of booster nodes for the first computing iteration as a function of the computation task, as taught by Lippert-Budenske. *Supra* Section VII.C.3.a). The POSITA would have done so in order to gain the additional advantages of Kambatla, namely to enable efficient scheduling of multiple tasks,

rather than one computational task, across the processing resources of the heterogeneous computing system, while minimizing "processing duration." EX1006, [0042], [0004], [0017], [0023]-[0024]; EX1003, ¶125.

b) Element 3.2

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious for the foregoing and following reasons. Lippert-Budenske-Kambatla includes an *application manager* that receives information from the nodes and processes the information to update an assignment of sub-tasks to the nodes. *Supra* Section VIII.B.1. Lippert-Budenske-Kambatla further includes a *resource manager*. *Supra* Section VIII.B.2.a). Lippert-Budenske includes a resource manager that dynamically alters further distribution of computation nodes and booster nodes during the computing of the computation task. *Supra* Section VII.C.3.b). Further, Kambatla discloses that its central resource manager includes various modules configured to handle different tasks and feed information to other modules "to aid in resource allocation." EX1006, [0036], [0031]-[0035]. Kambatla also discloses that "more or fewer components, organized in various configurations … remain[] within the scope" of Kambatla. *Id.*, [0030]. A POSITA thus would have naturally implemented Lippert-Budenske-Kambatla with a central resource manager as taught by Kambatla that includes a module configured to perform the dynamic-assignment functions of Lippert-Budenske's resource manager, thus achieving the benefits of

Kambatla's resource manager and the dynamic assignment of Lippert-Budenske's resource manager. *Supra* Sections VII.C.3.b), VIII.B. So implemented, the resource manager of Lippert-Budenske-Kambatla uses the information input by the application manager to dynamically alter further distribution during the computing of the computation task, thus combining the functions of the resource manager of Lippert, the HC Kernel Monitor and Basic Kernel of Budenske, and the resource manager of Kambatla. EX1003, ¶126.

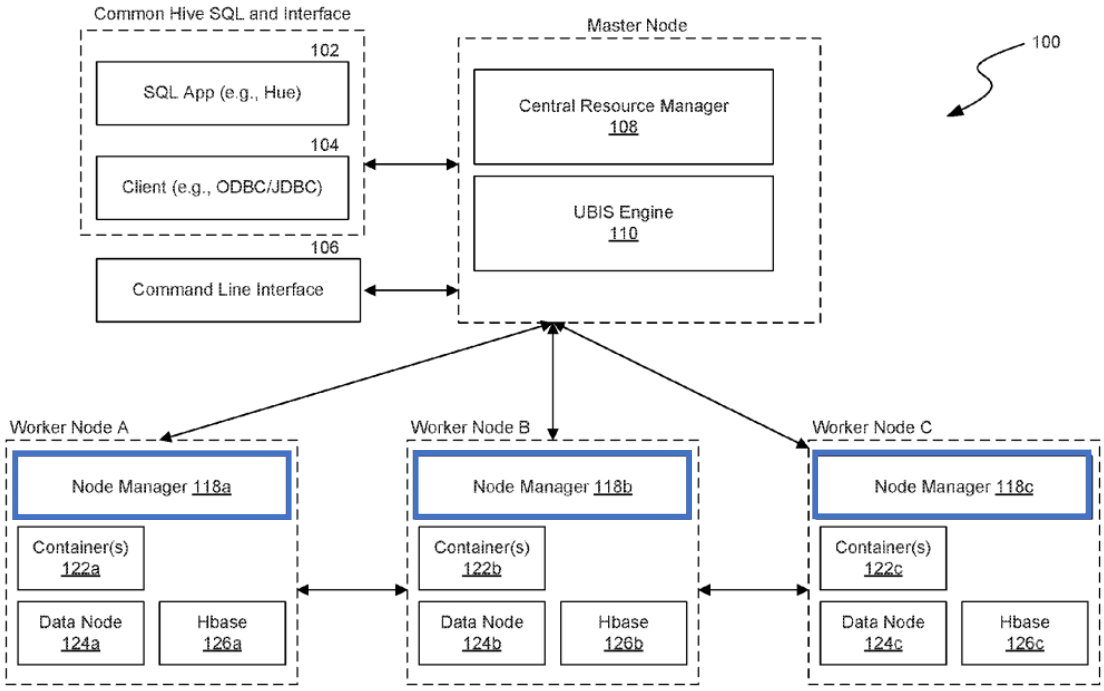
3. Claim 4

Lippert-Budenske-Kambatla teaches and discloses this element, and renders this claim obvious, at least because Lippert-Budenske teaches and discloses a resource manager that receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. *Supra* Sections VIII.B.2, VII.C.4. As Lippert-Budenske-Kambatla includes a resource manager that combines the central resource manager of Kambatla with the resource manager of Lippert-Budenske, the resource manager of Lippert-Budenske-Kambatla receives the information and dynamically alters assignment of the plurality of computation nodes and the plurality of booster nodes to each other during the computation of the computation task. EX1003, ¶127.

4. Claim 5

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. Lippert-Budenske renders claim 1 obvious. *Supra* Section VII.C.1. Further, Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.5. Moreover, Kambatla provides further implementation details by expressly disclosing a "resource manager daemon installed at worker nodes in a distributed computing cluster." EX1006, [0037]. The "resource manager daemon installed at the worker nodes includes a node manager 310b, and one or more application masters 320b." *Id.*, [0038], FIGS. 1, 3B. The central resource manager "[c]ommunicat[es] with node managers 118a-c which act as the agents at each

node," in order to "allocate and schedule resources available at the various nodes based on the available resources reported from each node manager." *Id.*, [0023].



Id., FIG. 1 (annotated). Each node manager "registers with the [central] resource manager and broadcasts the status of the node including the status of available resources … at the node." *Id.*, [0039]. Further, "[a]pplication masters 320b are responsible for requesting resources from the resource manager 108/110, working with node managers 310b to manage and monitor the containers 330b allocated by the resource manager 108/110." *Id.*, [0041]. Containers are an abstraction of the computing resources of the worker node that are allocated to process a task. *Id.*, [0004], [0023]; EX1003, ¶128.

In combination, Lippert-Budenske-Kambatla includes daemons as taught by Kambatla at each worker node—i.e., at each computation node and each booster node—in order to generate dynamic parameter information as taught by Budenske, as well as information regarding computing capacities and computation task requirements as taught by Lippert. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way because doing so would facilitate efficient communication of the relevant information in a status update for dynamically remapping the sub-tasks among the nodes, while the "additional information [taught by Budenske and Lippert] will typically add little overhead to an existing node [status update]." EX1006, [0039]; EX1003, ¶129.

5. Claim 8

Lippert-Budenske-Kambatla discloses and teaches this element, and renders this claim obvious. *Supra* Section VIII.B.2 (claim 3). Lippert-Budenske teaches and discloses this element. *Supra* Section VII.C.8. Further, according to Kambatla, the daemon operating at a node "monitors the containers 330b at the node … for resource utilization while processes are running" and "calculate[s] the aggregate resource utilization across all containers in the worker node" (*generates a measure of a loading of the node during processing*). EX1006, [0038]-[0040]; EX1003, ¶130.

In combination, Lippert-Budenske-Kambatla includes a daemon operating at each computation node and booster node that generates a measure of a loading of

the node during processing of a sub-task of the plurality of sub-tasks. A POSITA would have been motivated to implement Lippert-Budenske-Kambatla in this way, because doing so would allow the dynamic remapping of sub-tasks to the heterogeneous computing system as taught by Budenske to also account for "load balancing" as taught by Lippert. EX1004, [0017]; EX1003, ¶131.

6. Claim 9

a) Element [9.1]

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert and Budenske each disclose a heterogeneous computing system. *Supra* Section VII.C.1.a); EX1003, ¶132.

b) Element [9.2]

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert discloses a plurality of computation nodes and a plurality of booster nodes for computing one or more tasks comprising multiple sub-tasks. *Supra* Section VII.C.1.a); EX1003, ¶133.

c) Element [9.3]

Lippert-Budenske-Kambatla discloses and teaches this element, at least because Lippert-Budenske discloses and teaches this element. *Supra* Section VII.C.2. Kambatla also discloses a "cluster of worker nodes in [] communication

(e.g., via computer network) with each other and one or more master nodes." EX1006, [0021]; EX1003, ¶134.

In combination, Lippert-Budenske-Kambatla includes a communication interface connecting the plurality of computation nodes with each other and the plurality of booster nodes. EX1003, ¶135.

d) Element [9.4]

Lippert-Budenske-Kambatla discloses and teaches this element. *Supra* Sections VII.C.1, VIII.B.1, VIII.B.2. EX1003, ¶136.

e) Element [9.5]

Lippert-Budenske-Kambatla discloses and teaches this element. *Supra* Sections VII.C.1, VIII.B.1, VIII.B.4. EX1003, ¶137.

7. Claim 10

Lippert-Budenske-Kambatla discloses and teaches this claim. *Supra* Sections VIII.B.6, VIII.B.3; EX1003, ¶138.

IX. NO OBJECTIVE INDICIA OF NON-OBVIOUSNESS

Petitioner is unaware of any objective indicia of non-obviousness that overcome the above grounds of unpatentability as to any challenged claim. Undersigned Counsel for Petitioner is aware that Patent Owner has made generalized and undeveloped allegations of secondary considerations in a confidential interrogatory response served in district court litigation. Given a protective order in

that district court litigation, however, IPR Counsel for Petitioner does not have access to those allegations of secondary considerations and therefore is unable to address them at this stage of the IPR proceedings. Should Patent Owner decide to present any arguments or alleged evidence of secondary considerations in this IPR, Petitioner reserves the right to address such arguments and evidence at the appropriate time. *See, e.g.*, *Microsoft Corp. v. Interdigital Patent Holdings, Inc.*, IPR2024-00305, Paper 12 at 23 (PTAB May 10, 2024).

X. CONCLUSION

Petitioner requests institution for claims 1-10, on Grounds 1 and 2 specified in this petition.

Respectfully submitted,

Dated: December 20, 2024

By: /Andrew M. Mason/

Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner

CERTIFICATE OF COMPLIANCE WITH
TYPE-VOLUME LIMITATION PURSUANT TO 37 C.F.R. § 42.24

This brief complies with the type-volume limitation of 37 C.F.R. § 42.24(a)(1)(i).

The brief contains **13,735** words, excluding the parts of the brief exempted by 37 C.F.R. § 42.24(a).

The brief has been prepared in a proportionally spaced typeface using Microsoft Word for O365 in a 14-point Times New Roman font.

Dated: December 20, 2024

By: /Andrew M. Mason/

Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner

CERTIFICATE OF SERVICE
IN COMPLIANCE WITH 37 C.F.R. § 42.6(E)(4)

The undersigned certifies that the **PETITION FOR *INTER PARTES* REVIEW OF U.S. PATENT NO. 11,537,442 and EXHIBITS 1001 – 1025** were served on December 20, 2024, via **Express Mail** on the Patent Owner at the following address of record as listed on the USPTO Patent Center:

Marshall, Gerstein & Borun LLP
233 South Wacker Drive
6300 Willis Tower
Chicago, Il 60606-6357

By: /Andrew M. Mason/
Andrew M. Mason (Reg. No. 64,034)
andrew.mason@klarquist.com
KLARQUIST SPARKMAN, LLP
One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Tel: 503-595-5300
Fax: 503-595-5301

Counsel for Petitioner

Available online at www.sciencedirect.com
SCIENCE DIRECT
J. Parallel Distrib. Comput. 66 (2006) 77–98
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

A semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems☆

Yu-Kwong Kwok[a], Anthony A. Maciejewski[b], Howard Jay Siegel[b,c], Ishfaq Ahmad[d,*], Arif Ghafoor[e]

[a]*Department of Electrical and Electronic Engineering, The University of Hong Kong, Pokfulam Road, Hong Kong, Hong Kong*
[b]*Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523-1373, USA*
[c]*Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1873, USA*
[d]*Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, TX 76019-0015, USA*
[e]*School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285, USA*

Received 17 February 2005; received in revised form 2 June 2005; accepted 23 June 2005
Available online 10 November 2005

Abstract

Minimization of the execution time of an iterative application in a heterogeneous parallel computing environment requires an appropriate mapping scheme for matching and scheduling the subtasks of a given application onto the processors. Often, some of the characteristics of the application subtasks are unknown a priori or change from iteration to iteration during execution-time based on the inputs being processed. In such a scenario, it may not be feasible to use the same off-line-derived mapping for each iteration of the application. One possibility is to employ a *semi-static* methodology that starts with an initial mapping but dynamically performs remapping between application iterations by observing the effects of the changing characteristics of the application's input data, called *dynamic parameters*, on the application's execution time. A contribution in this paper is to implement and evaluate a semi-static methodology involving the on-line use of off-line-derived mappings. The off-line phase is based on a genetic algorithm (GA) to generate high-quality mappings for a range of values for the dynamic parameters. A dynamic parameter space partitioning and sampling scheme is proposed that partitions the parameter space into a number of *hyper-rectangles*, within which the "best" mapping for each hyper-rectangle is stored in a mapping table. During the on-line phase, the actual dynamic parameters are observed and the off-line-derived mapping table is referenced to choose the most suitable mapping. Experimental results indicate that the semi-static approach outperforms a dynamic on-line approach and performs reasonably close to an infeasible on-line GA approach. Furthermore, the semi-static approach considerably outperforms the method of using the same mapping for all iterations.
© 2005 Elsevier Inc. All rights reserved.

Keywords: Heterogeneous computing; Parallel processing; Mapping; Scheduling; Iterative task graphs; Genetic algorithms; Automatic target recognition

1. Introduction

1.1. On-line use of off-line-derived mappings

Heterogeneous computing (HC) encompasses a great variety of situations (e.g., see [19,24,28,35,46,48,53,54]). This paper focuses on a particular application domain in which (1) an iterative application is to be mapped onto an associated specific type of dedicated heterogeneous parallel hardware platform and (2) the execution of each iteration can be represented by a directed acyclic graph (DAG) of

☆ This research was jointly supported by the Hong Kong Research Grants Council (under Contract No. HKU 7124/99E), by the DARPA/ITO Quorum Program (under NPS Subcontract Nos. N62271-98-M-0217 and N62271-98-M-0448, and GSA Contract No. GS09K99BH0250), by the DARPA/ITO Quorum Program through the Office of Naval Research (under Grant No. N00014-00-1-0599), and by the Colorado State University Abell Endowment.
* Corresponding author. Fax: +1 817 272 3784.
E-mail addresses: ykwok@eee.hku.hk (Yu-Kwong Kwok), aam@colostate.edu (A.A. Maciejewski), hj@colostate.edu (H.J. Siegel), iahmad@cse.uta.edu (I. Ahmad), ghafoor@ecn.purdue.edu (A. Ghafoor).

0743-7315/$ - see front matter © 2005 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2005.06.015

Darell Long, Ph.D.
09/17/2025
Exhibit G
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1007

ParTec EX2022
Microsoft v ParTec, IPR2025-00318

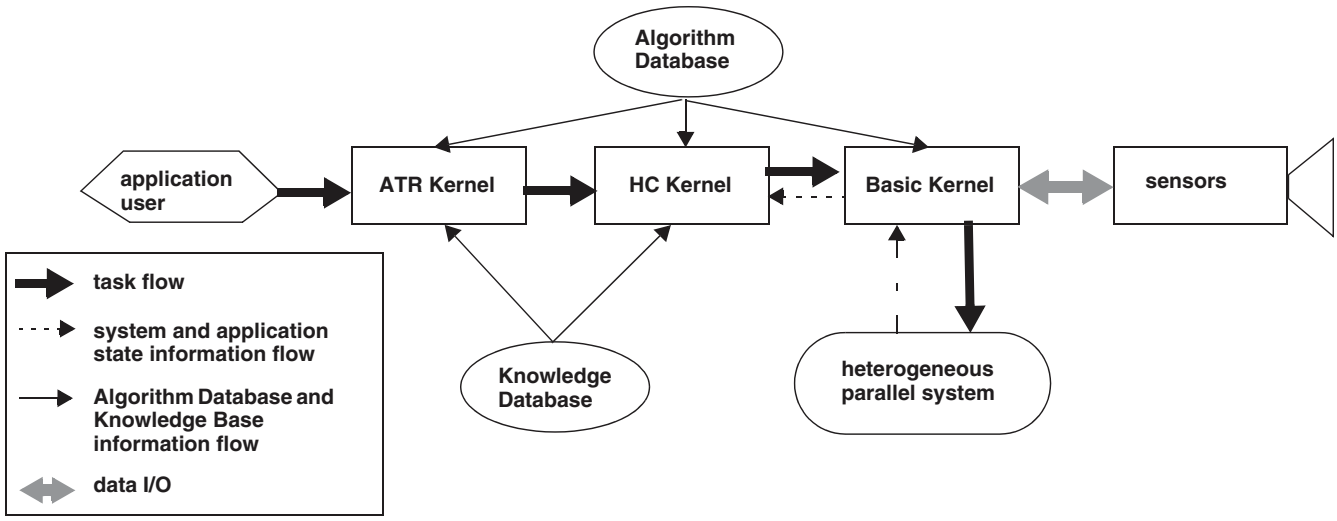


Fig. 1. Conceptual design of an operating system for ATR applications.

subtasks. To minimize the execution time of such an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed for matching and scheduling the subtasks onto the processors [1,2,5–8,29,39,34,55]. However, when some of the characteristics of the application subtasks are unknown a priori and will change from iteration to iteration during execution-time based on the inputs being processed, it may not be feasible or desirable to use the same off-line-derived mapping for each iteration of the application.

An example of such a problem domain are iterative automatic target recognition (ATR) tasks, where a sequence of images are received from a group of sensors and various image-processing operations are required to generate a real-time scene description. Most notable ATR-based applications are homeland security imaging [50] and medical monitoring systems [32]. In ATR, the characteristics of a subtask's input data, such as the amount of clutter and the number of objects to be identified, varies from image to image, and thus, may lead to large variations in the subtask's processing requirements from one iteration to the next.

In such situations, a semi-static methodology [9,10] can be employed, which starts with an initial mapping but dynamically decides whether to remap the application with a mapping previously determined off-line (i.e., statically). This can be done by observing, from one iteration to another, the effects of the changing characteristics of the application's input data, called dynamic parameters, on the application's execution time. In other words, the operating system will be able to make a heuristically determined decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system will be able to select a pre-computed and stored mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking). This remapping process will, in general, require a certain system reconfiguration time for relocating the data and program modules. The semi-static method differs considerably from other real-time HC mapping techniques in that it involves the on-line real-time use of off-line precomputed mappings. This is significant because it is possible for off-line heuristics to have much longer execution times to search for a good solution (mapping) than what is practical for an on-line heuristic. Thus, with the semi-static method, the mapping quality of a time-consuming off-line heuristic can be approached at real-time speeds. The focus of this paper is the evaluation of a method for performing semi-static mappings.

A previously proposed conceptual design of a high-level operating system for ATR applications, which includes the capability for the on-line use of off-line computed mappings, is depicted in Fig. 1 [9,10]. This conceptual design has its roots in the high-level model presented in [14] for automatic dynamic processor allocation in a partitionable parallel machine with homogeneous processors (called PASM [47,49]).

The ATR Kernel in Fig. 1 makes decisions on how a given ATR application task should be accomplished, including determining the partial ordering of subtasks and which algorithms should be used to accomplish each subtask. The HC Kernel uses a semi-static method to decide how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. A subtask may have a data-parallel implementation, and, thus, may be assigned to a set of processors. Also, the HC Kernel interacts with the Basic Kernel (the low-level operating system) to execute the application and monitors its execution so it can decide at the end of each iteration through the application if the subtasks should be remapped onto the hardware platform. Thus, the ATR Kernel deals with application issues, while the HC Kernel deals with implementation issues. Information from the Algorithm Database and

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 303 of 411

the Knowledge Base is used to support the ATR and HC Kernels.

The design of the ATR Kernel and HC Kernel was the focus of Budenske et al. [10]. In addition, two methods were suggested for determining the representative choices for the dynamic parameter values to use for each static mapping. The first was to have the application developer specify what these representative values should be. The second was to have the application developer specify a range for each dynamic parameter, and then the representative values are uniformly distributed over that range. Neither of these methods was ever evaluated in any way, as this was not the focus of Budenske et al. [10]. In particular, the performance of off-line mappings based on points selected in either of these ways was never determined. In contrast, our current work focuses just on the semi-static mapping approach. The current research does not depend on an application developer to guess at what might be appropriate representative values; nor does it take the simplistic approach of selecting representative values uniformly across the range. Instead, we derive a procedure for statistically sampling each region of the multidimensional dynamic parameter space to find a mapping that is near optimal for the range of parameters within that region. We evaluate the current approach in a variety of ways, including comparing it to the simplistic uniform distribution method of Budenske et al. [10].

A more detailed discussion of the ATR problem domain is presented in Section 1.2. Examples of other applications whose characteristics are similar to those of the iterative ATR applications and platforms include sensor-based robotics, intelligent vehicle highway systems, air traffic control, nuclear facility maintenance, weather prediction, intruder detection, and manufacturing inspection. The performance of the semi-static method in comparison to other methods will depend upon the exact application and exact platform under consideration.

The application to be mapped is iterative in structure and each iteration is modeled by a DAG in which the nodes represent subtasks and the edges represent the communications among subtasks. The model used for an application task is described in Section 3. The attributes associated with the DAG, such as the computation time of a subtask and the communication time between subtasks, are modeled by equations that are functions of the dynamic parameters. Examples of dynamic parameters include the contrast level of an image, the number of objects in a scene, and the average size of an object in a scene. Thus, as the dynamic parameters change from one iteration (one image) to the next iteration, the mapping currently in use may not be suitable and a remapping of the subtasks onto the processors may need to be performed. However, performing a remapping requires a certain system reconfiguration time. Given the current mapping, a new mapping, and the system estimated reconfiguration time, the HC Kernel has to decide whether a remapping is to be done. This framework can be applied to any task graph structure represented as a DAG.

1.2. An example application domain: ATR

Simply stated, an ATR system takes a set of images iteratively from a group of sensors and produces some type of description of the scene [56]. The most notable real-life examples of an ATR-based system are homeland security imaging [50] and medical monitoring systems [32]. A simplified example of an ATR task for tracking forwardlooking ladar infra-red (FLIR) images is shown in Fig. 2 [59]. The various types of image processing elements required in an ATR system can be broadly classified into three groups: low-level processing (numeric computation), intermediate-level processing (quasi-symbolic computation, e.g., where numeric and symbolic types of operations are used to describe surfaces and shapes of objects in the scene), and high-level processing (symbolic computation, e.g., for producing the scene description) [3,4,13,15,20,25,27]. Heterogeneous parallel architectures are appropriate computing platforms for efficiently handling computational tasks with such diverse requirements.

A key technical issue that must be addressed to exploit the inherent potential of heterogeneous parallel computing systems to efficiently execute ATR applications is the development of a high-level operating system that can fully utilize the architectural flexibility of such a system. Such a high-level operating system must be able to assign each ATR application subtask to the processors where it is best suited for execution. Often, subtasks can execute concurrently, sharing resources. Because the execution time of application subtasks in an ATR system is highly input-data dependent (e.g., number of currently located objects), this matching and scheduling of application subtasks to processors must be performed dynamically at execution-time for best performance.

The semi-static concept was proposed in [10] for a class of ATR applications, where each application can be modeled as an iterative execution of a set of partially ordered subtasks. The ATR applications in this class are production jobs that are executed repeatedly and, hence, it is worthwhile to invest off-line time to determine an effective mapping of such an application onto the hardware platform used to execute it. The automatic target acquisition (ATA) system described in [16] and the tracking system described in [17] are examples of such iterative ATR applications.

1.3. Contributions of this research

The objective of this paper is to implement and evaluate a semi-static methodology, called the on-line use of off-line-derived mappings (denoted as On–Off in subsequent sections), whose conceptual structure was proposed in [10]. In particular, the goal of the present study is two-fold: (1) to design novel and practical off-line mapping generation methods; and (2) to evaluate the ideas underlying the design and use of the On–Off method for a specific class of

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 304 of 411

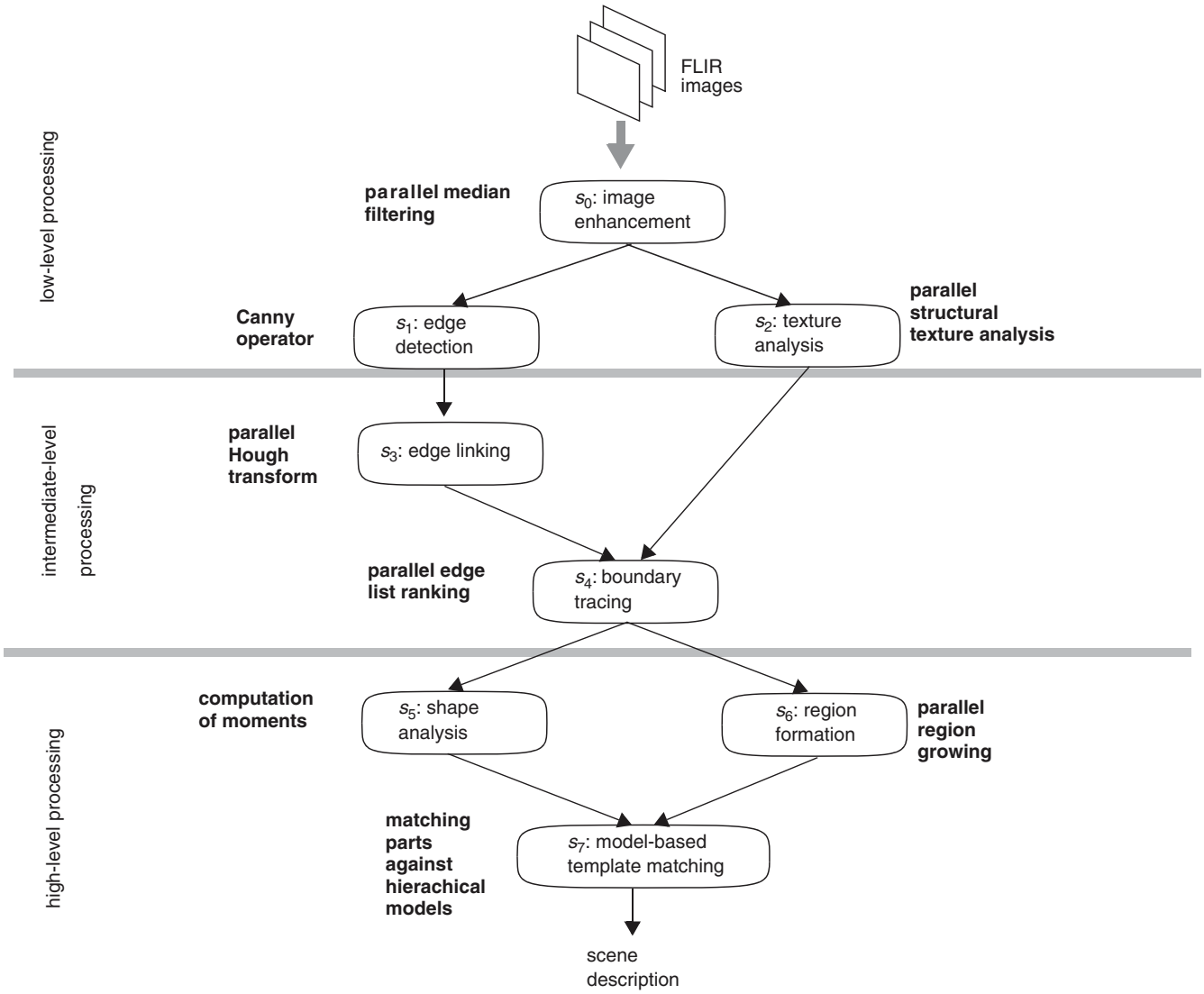


Fig. 2. A directed acyclic graph modeling an ATR application.

computational structures and hardware platforms. This is done by simulating the methodology and studying its behavior for various situations.

The implementation of the On–Off methodology entails addressing the fundamental research issue of how to select representative mappings off-line for on-line use. To solve this problem, a novel dynamic parameter space partitioning and sampling scheme is proposed in Section 4. During the off-line phase, a genetic algorithm (GA) is used to generate high-quality mappings for a range of values for the dynamic parameters. Specifically, the dynamic parameter space is partitioned into a number of hyper-rectangles, within which the "best" mapping for each hyper-rectangle is stored in a mapping table. During the on-line phase, the actual dynamic parameters are observed and the off-line-derived mapping table is referenced to choose the most suitable mapping. Experimental results, presented in Section 5, indicate that this semi-static approach is effective in that it consistently gave performance that was comparable to that of using the same GA on-line with the exact dynamic parameter values for the *next* iteration (which is physically impossible). Also, the semi-static approach considerably outperforms using the same mapping for all iterations and outperforms a representative dynamic on-line mapping heuristic.

2. Related work

In [38], a greedy policy is suggested for the dynamic remapping of iterative data parallel applications, such as fluid dynamics problems, on a homogeneous message-passing architecture. In these types of applications, multiple processors work independently on different regions of the data domain during one iteration. As the focus of computation may shift from one region to another between iterations, some processors will be idle while some other processors

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 305 of 411

will be overloaded if the initial mapping of workload is not changed. Thus, remapping is useful for balancing the workload across the processors and thus, reducing the execution time of an iteration. The technique reported in [38] works by monitoring from iteration to iteration a function that indicates the average processor idle time since the last remapping. Once it is detected that a local minimum point of this function is reached at a certain iteration, a remapping is done. This approach is not applicable to the scenario considered in this work for two reasons. First, it assumes a homogeneous processing environment rather than a heterogeneous one. Second, it is designed for independent data parallel computations such that a task does not have the subtask precedence constraints that are modeled as DAGs here.

A framework has been recently suggested for determining an off-line mapping of image processing applications, modeled as a linear task chain, to homogeneous distributed-memory machines [31]. The mapping technique, which is based on the shortest path algorithm, can only be applied to a linear task chain. The goal is to produce a static mapping that has good average performance without regard to the actual variations of task parameters during execution time. This is in contrast to the semi-static technique described here that adjusts the mapping during execution time according to the changes in dynamic parameters. The algorithm used in [31] is not applicable to the scenario considered here because tasks are represented as DAGs, rather than chains, and the hardware target is assumed to be heterogeneous rather than homogeneous.

A scheduling and allocation scheme reported in [41] for regular scientific applications uses a model that is similar to the one used here in that a subtask in an application is a data parallel program. According to this model, a regularly structured high-performance fortran (HPF) application is decomposed into a DAG in which the functional parallelism is captured by the DAG structure but the data parallelism is encapsulated within each subtask. An approximate algorithm is used for mapping the application to a distributed-memory machine. However, the approach is static in that the mapping algorithm is applied based on cost estimates at compile-time and no execution-time scheme is used for tuning the scheduling and allocation. Furthermore, a homogeneous target platform is assumed.

In [60], a run-time support technique is proposed for minimizing the execution time of some iterative computations on message-passing architectures by using the loop unfolding method. The technique used is essentially based on exploiting the inter-iteration parallelism to further minimize the execution time. Similar to the above approaches, the mapping is only done statically based on cost estimates and no execution-time adjustment of mapping is done, and a homogeneous target platform is assumed.

In [52], an automatic machine selection scheme is proposed for allocating application tasks to a network of autonomous computers. The major focus is to minimize network congestion by taking into account the application's communication topology. Unlike the task-mapping algorithms considered in this paper, execution times of the applications are not explicitly optimized. Several scientific applications (e.g., 2D fast-Fourier transform) are tested using an experimental system implemented using the Remos API [42]. In [45], the dynamic data redistribution costs of the 2D Jacobi application are analytically modeled by exploiting the application's structure under the AppLeS (Application Level Scheduling) framework. The model is rather specific and can accurately predict the redistribution costs of the 2D Jacobi application under different simulated scenarios. This is very different from the generalized task execution model used in the study presented here.

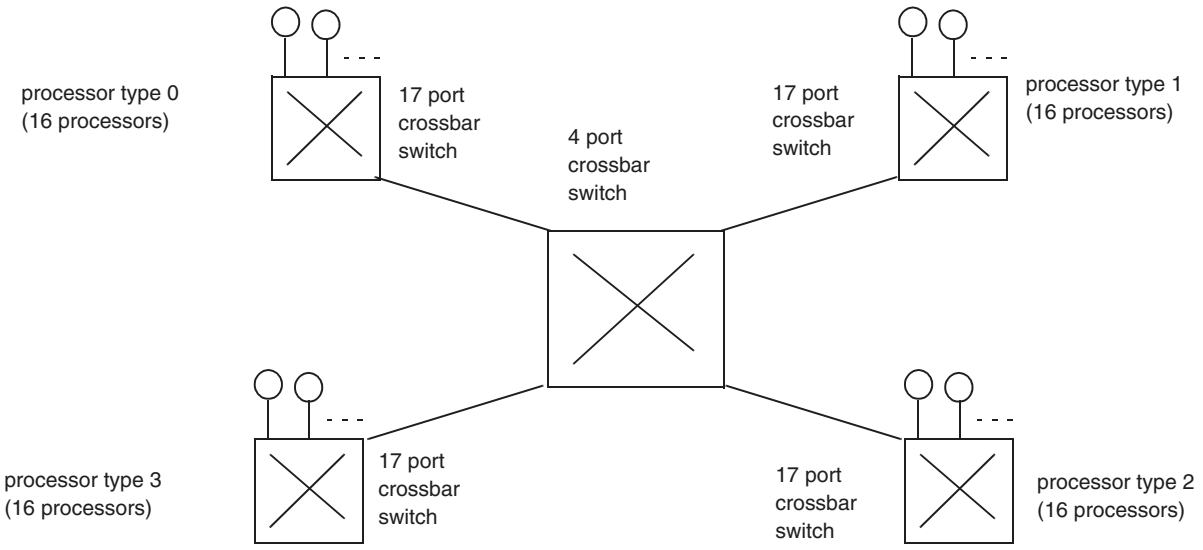
In [43], an integrated compile-time and run-time scheme for predicting the data redistribution cost for multi-phase parallel applications on a software distributed shared memory system is suggested. The crux of the scheme is a clever exploitation of the knowledge about the data access patterns, page locations, and the distributed shared memory protocol. Again, the focus is mainly on execution costs related to data placements but not the overall execution times. In [37], an integrated load balancing and scheduling scheme is proposed for efficient execution of parallel applications on a network of autonomous machines. The proposed scheme exploits the synergy among the compile-time analysis mechanism, run-time system management, and OS level load balancing facilities. Unlike the general HC platform considered here, the target platform is a loosely coupled homogeneous machines. On a network of four machines, each of which is a four-CPU SMP, the integrated scheme can efficiently execute several scientific applications (e.g., Jacobi, matrix multiplication, etc.).

3. System model

To evaluate the On–Off semi-static mapping methodology, a particular sample architecture is chosen; however, the On–Off method can be adopted for other target architectures. The sample target HC platform considered here is based on the expected needs of ATR applications that are of interest to the US Army Research Laboratory (e.g., [16,17]). Specifically, it contains four different types of processors (e.g., [18,30]), with 16 processors of each type (see Fig. 3(a)).

The processors are connected via crossbar switches in such a way that each processor has exactly one input port and one output port. Communications among processors of the same type are assumed to be symmetric in the sense that the conflict-free time for any pair of processors (of the same type) to communicate is the same (see Fig. 3(b)). For simplicity, it is assumed that if a data-parallel implementation of a given subtask uses a virtual machine of processors, all processors will be of the same type. Given this and the symmetry property of the inter-processor communications among processors of the same type, the expected execution

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 306 of 411



(a)

	type 0		type 1		type 2		type 3	
	S	1/*R*	*S*	1/*R*	*S*	1/*R*	*S*	1/*R*
type 0	0.92	0.41	9.04	3.72	10.47	5.58	22.47	11.16
type 1	9.04	3.72	0.83	0.37	4.62	4.40	7.00	7.44
type 2	10.47	5.58	4.62	4.40	0.86	0.45	11.37	6.38
type 3	22.47	11.16	7.00	7.44	11.37	6.38	0.83	0.53

(b)

Fig. 3. (a) The target heterogeneous computing platform consisting of four types of processors with 16 in each type; (b) message start-up times, S (in ms), and transmission times per unit data, $1/R$, of the inter-processor communication channels.

time of a particular multiprocessor implementation of a subtask is independent of which fixed-size subset of the processors of a given type are assigned to execute the subtask. It is assumed that one processor in each virtual machine is responsible for data communication to other virtual machines. This assumption simplifies the simulation study, but is not required for the On–Off methodology. In addition, the hardware platform includes (1) a workstation, for off-line operations to develop an application implementation and for use as the application user interface, and (2) a host processor, which monitors the application implementation during its execution and implements the on-line HC Kernel. It is assumed that when a given ATR application is executing on a platform, that platform is dedicated to that application. It should be noted that the On–Off strategy is applicable to a wide variety of computing platforms that vary from the test platform described.

An application task is modeled as a DAG, with n nodes representing subtasks s_i $(0 \leqslant i \leqslant n-1)$ and e edges representing inter-subtask communications. To illustrate the On–Off semi-static mapping approach, a simplified model is used for subtask execution time and inter-subtask communication time. However, the On–Off framework does not depend on how the subtask execution time and inter-subtask communication time are modeled. The way in which the four dynamic parameters α, β, γ, and μ are used in each subtask execution time (and inter-subtask communication time) equation given below is a simple approach for the purpose of illustration in the simulation studies to be described. In reality, a single-dynamic parameter can impact any subset of the components of a given subtask's execution time equation. Furthermore, the way in which a particular dynamic parameter impacts a given component of a subtask's execution time may not be linear (as assumed here), and may differ for different subtasks. Clearly, the structure and details of the execution time complexity equations for subtasks, as a function of dynamic parameters, is task dependent and can vary greatly. The determination of these equations is considered the responsibility of the application developer [9,10] and outside the scope of this paper. These equations are just an input to the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 307 of 411

Table 1
Definition of notation

Notation	Definition
s_i	Subtask i
n	Number of subtasks in the application task
Subtask execution time equation variables	
$t_u(s_i)$	Execution time of subtask s_i on virtual machine u
h_{iu}	Heterogeneity factor indicating the relative speed of the ith subtask on machine u
a_i	Coefficient for the parallel portion of subtask i's execution time
b_i	Coefficient for the parallelization overhead portion of subtask i's execution time
c_i	Coefficient for the serial portion of subtask i's execution time
p	Number of processors used in a virtual machine
Inter-subtask communication time equation variables	
d_{ij}	Size of the fixed portion of the data to be transferred between subtasks s_i and s_j
e_{ij}	Coefficient for the size of the variable portion of data to be transferred between s_i and s_j
C_{uv}	Communication time between virtual machines u and v
S_{uv}	Message start-up time between virtual machines u and v
R_{uv}	Data-transfer rate between virtual machines u and v
Dynamic parameters	
α	Dynamic parameter impacting the parallel workload
β	Dynamic parameter impacting the parallelization overhead
γ	Dynamic parameter impacting the serial workload
μ	Dynamic parameter impacting the size of the variable portion of data transfer

On–Off methodology presented here, and when considering the results of the paper the reader should note that the execution time and communication time equations are parts of a simplified model. Table 1 summarizes the notation to be used throughout the paper.

The simple execution time expression used in this model is a version of Amdahl's law extended by a term representing the parallelization overhead (e.g., synchronization and communication). The serial and parallel fractions of a subtask are frequently represented using similar models (e.g., [11,21,36,44]). The execution time expression for subtask s_i includes: (a) three dynamic parameters, α, β, and γ, (b) the number of processors used, p, and (c) three coefficients a_i, b_i, and c_i. The parallel fraction and serial fraction of subtask s_i are represented by $a_i\alpha/p$ and $c_i\gamma$, respectively. The parallelization overhead is represented by $b_i\beta\log p$ and h_{iu} is the heterogeneity factor, indicating the relative speed of the subtask s_i on the type of processor used in virtual machine u. The heterogeneity factor reflects the real-world differences among machines that could impact the execution time of a given subtask on a given machine. These factors include, but are not limited to, CPU clock rate, the number of levels of cache and sizes of cache at each level, the exact instruction set and execution time for each instruction, and the pipeline structure. The impact of each of such factors will depend on the precise code for the given subtask. Thus, in general, different subtasks will have affinities for different machines. The execution time $t_u(s_i)$ of subtask s_i on virtual machine u is modeled by the expression

$$t_u(s_i) = h_{iu} \cdot (a_i\alpha/p + b_i\beta\log p + c_i\gamma). \tag{1}$$

By differentiating this equation and equating the derivative to zero, the optimal value of p that leads to the minimum execution time for a given subtask is $p_{\text{opt}} = (a_i\alpha)/(b_i\beta)$. The mapping heuristic will not assign more processors than a subtask's p_{opt}.

It is assumed that the size of the data set to be transferred between two subtasks s_i and s_j consists of a fixed portion and a variable portion. The size of the fixed portion is modeled by a constant d_{ij} (independent of the input data of the application). The size of the variable portion is modeled by the product of a coefficient e_{ij} and a dynamic parameter μ. For communication between virtual machines u and v, S_{uv} and R_{uv} are the message start-up time and the data transmission rate, respectively [26] (see Fig. 3(b) for values of S_{uv} and R_{uv} based on [17,18,26]). Thus, the inter-subtask communication time between subtask s_i on virtual machine u and subtask s_j on virtual machine v is C_{uv}, which is given by

$$C_{uv}(s_i, s_j) = S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}. \tag{2}$$

4. The semi-static mapping approach

4.1. Remapping approaches

Consider two approaches for remapping application tasks to processors during execution time (between iterations through the DAG):

- *Dynamic mapping*: Based on the current values of dynamic parameters, compute a new mapping in real time using a low complexity algorithm.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 308 of 411

- *On-line use of off-line-derived mappings*: For each dynamic parameter, some representative values are chosen so that a number of possible scenarios are generated. Using an off-line (i.e., static) heuristic, high-quality mappings for the scenarios are precomputed and stored in a table. During execution of the application, the mapping corresponding to the scenario with values of dynamic parameters *closest to* the actual values is selected from the table to be a possible new mapping [5].

Because a static mapping heuristic (e.g., the GA used in this study) can potentially generate solutions of much higher quality than a dynamic mapping algorithm, it is interesting to investigate how well the approach of on-line use of off-line-derived mappings (using the GA) performs. Notice that even off-line generation of optimal mappings is infeasible because the heterogeneous mapping problem is NP-complete [22,23], and, thus, exponential time is needed for finding optimal solutions.

4.2. Generation of off-line-derived mappings

In the On–Off semi-static mapping approach, it is assumed that the ranges of the dynamic parameters are known. This assumption is justified because, for example, for a particular size of image, the maximum possible number of objects of a given type at a given distance is bounded and can be estimated. Once specified by the application developer, the space of dynamic parameters is partitioned into a number of disjoint regions. Formally, suppose the minima and total range sizes of the dynamic parameters are given by $\alpha_{\min}, \beta_{\min}, \gamma_{\min}, \mu_{\min}, \alpha_{\text{range}}, \beta_{\text{range}}, \gamma_{\text{range}}$, and μ_{range}, respectively. The parameter space $\Re$ can be partitioned into K^4 uniform sized disjoint regions as follows:

$$\Re(i, j, k, l) = \{(\alpha, \beta, \gamma, \mu)\}, \quad 0 \leqslant i, j, k, l \leqslant K-1, \quad (3)$$

where $\alpha_{\min} + i\alpha_{\text{range}}/K \leqslant \alpha < \alpha_{\min} + (i+1)\alpha_{\text{range}}/K$ and the ranges for β, γ, and μ are defined analogously.

Within each region (defined by specifying values for indices i, j, k, l), N dynamic parameter vectors, each composed of $(\alpha, \beta, \gamma, \mu)$, are randomly chosen. An off-line (static) heuristic is then applied to determine the mappings for these sample scenarios represented by different dynamic parameter vectors. For a random sample vector v_x $(0 \leqslant x \leqslant N-1)$, denote the corresponding mapping by M_x. The mapping M_x is then evaluated for every one of the $N-1$ other sample scenarios in the region by applying the mapping M_x to the DAG and computing the total completion time for one iteration of the application given the dynamic parameter values that define that scenario. This is repeated for each M_x. That is, the task completion times $t(M_x(v_y))$ for all x and y $(0 \leqslant x, y \leqslant N-1)$ are computed. Then, the average completion time for each mapping M_x is computed as $[\sum_{y=0}^{N-1} t(M_x(v_y))]/N$. The mapping M_x that gives the minimum of these average completion times is chosen as the representative mapping for the corresponding region in the dynamic parameter space. This representative mapping and the corresponding average completion time are stored in the off-line mapping table, which is a multi-dimensional array indexed by i, j, k, l.

This new approach for determining representative mappings differs considerably from that used in earlier work, where it was assumed that the application developer will provide the specific set of dynamic parameter values to be used to derive each representative mapping [10]. Fig. 4 depicts the block diagram of the envisioned system for determining the off-line mapping for a given region of the dynamic parameter space.

Here, the parameter space is uniformly partitioned with each parameter equally subdivided. However, different values of K can be used for each individual dynamic parameter, depending upon the specifications given by the application developer. In addition, one could use non-uniform-sized regions for the parameter space if good performance is not being achieved. In particular, if the average completion time for the representative mapping selected for a region is significantly greater than $t(M_x(v_x))$ for most values of x, then that region can be subdivided and a representative mapping determined for each sub-region. This process can be repeated recursively.

4.3. On-line retrieval of off-line-derived mappings

The input to the simulated on-line module consists of an execution profile that comprises a certain number of iterations of executing the task graph. Examples of execution profiles containing 20 iterations are shown in Table 2 in Section 5. In each profile, the dynamic parameter values change from one iteration to another. Specifically, row i represents the values of the dynamic parameters *observed* after execution of the graph for iteration i is finished. Thus, when execution of the task for iteration i begins, the on-line module does not know the (simulated) *actual* values of the dynamic parameters for that iteration. The on-line module has to determine a mapping for iteration i based on the dynamic parameter values of iteration $i-1$. In the On–Off semi-static mapping approach, the representative mapping is retrieved for the region that includes the dynamic parameter values of iteration $i-1$. If the stored pre-determined execution time of the selected mapping, plus the estimated reconfiguration time, is smaller than the execution time that occurred for iteration $i-1$, a remapping is performed; otherwise, the mapping used at iteration $i-1$ will continue to be used for iteration i. The off-line mapping technique used in this study is discussed in Section 4.5.

4.4. A dynamic approach

In this paper, several mapping approaches are examined and evaluated. The first approach is a dynamic approach that uses a fast heuristic that takes a small amount of time but generates a reasonably good solution. The heuristic used is a

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 309 of 411

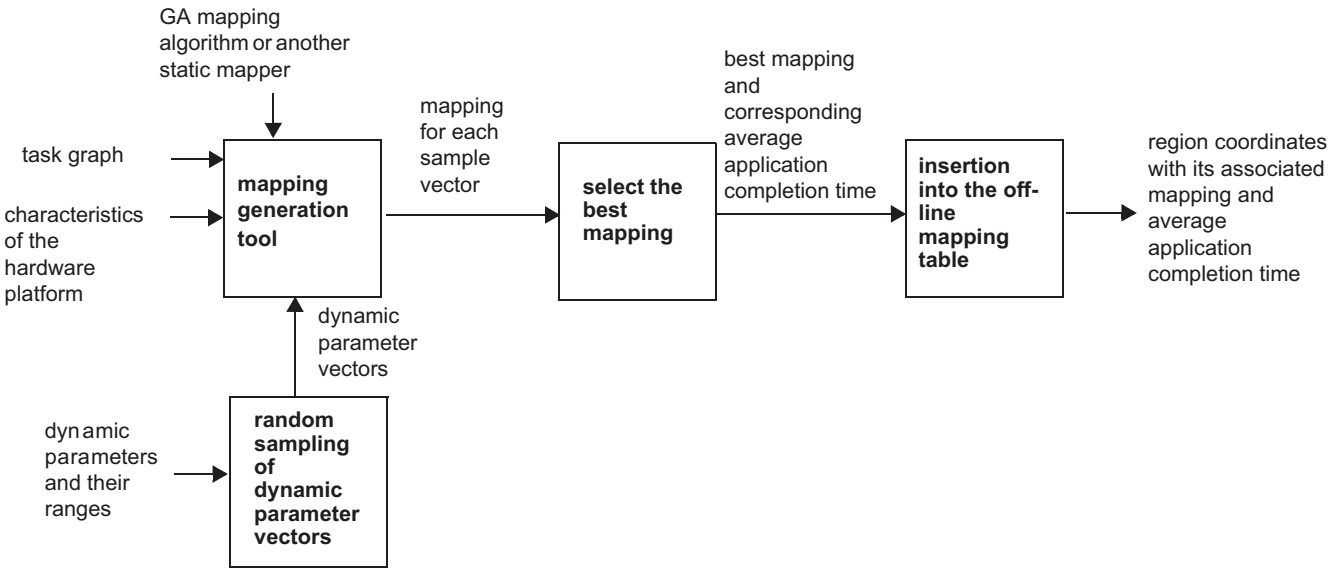


Fig. 4. Generation of the off-line mapping for a given region of the dynamic parameter space.

fast static mapping algorithm, called the earliest completion time (ECT) algorithm, that is based on the technique presented in [57] and is outlined below. It should be noted that other high-performance scheduling heuristics, such as DCP [29] and CPR [40], are unsuitable to be used for dynamic scheduling due to their high time complexity.

The ECT algorithm is used for performing dynamic mapping in a manner similar to that described in Section 4.3. The ECT algorithm is applied (in real time) to the task graph with the values for the dynamic parameters at iteration $i-1$. The resulting mapping is (with its associated estimated task execution time using the iteration $i-1$ parameters) then considered to be a potential new mapping for iteration i. Again, if the estimated gain in adopting the new mapping is greater than the reconfiguration time, the new mapping will be used in iteration i.

ECT ALGORITHM:

(1) Construct a level-based list L of subtasks as follows:
  (a) Label each entry subtask (subtask that does not have predecessor) as a level 1 subtask.
  (b) For each of the remaining subtasks, label it as level i if its highest level parent is at level $i-1$.
  (c) Make L a list of subtasks sorted in ascending order of levels. For subtasks on the same level, they are sorted in descending order of the number of children. Ties are broken arbitrarily.
(2) for each subtask s_i on L do:
(3) for each processor group do:
(4) for $p = 1$ to p_{opt} do:
(5) Find the earliest start time t_{start} such that there are p available processors.
(6) Note s_i's completion time using p processors.
(7) endfor
(8) endfor
(9) Schedule s_i using p processors of the processor group (starting at the corresponding t_{start}) that minimizes its completion time.
(10) Update the availability times for the processors in this group.
(11) endfor

4.5. Genetic algorithm

GAs are a promising heuristic approach to tackling optimization problems that are intractable. There are a great variety of approaches to GAs (see [51] for a survey). The first step of designing a GA is to encode all possible solutions as chromosomes, a set of which is referred to as a population. In this study, the GA presented in [58] is extended to allow multiple processors to be assigned to a subtask. Each chromosome consists of three parts: the matching string, the processor allocation string, and the scheduling string. Each string is a vector of length n, the number of subtasks in the application task.

Let *mat* be the matching string where $mat(i) = j$ means that subtask s_i is mapped to processor type j. Let the allocation string, $\text{alloc}(i)$, be the number of processors of type

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 310 of 411

Table 2
Execution profiles of dynamic parameters: (a) Profile A (average percentage change in dynamic parameter values $\Delta = 5\%$) and (b) Profile B ($\Delta = 40\%$)

Iteration	(a) Profile A			
	α	β	γ	μ
0	3000	15	300	60
1	2821	15	287	63
2	2949	12	302	65
3	3073	12	286	68
4	3228	11	273	71
5	3090	13	258	67
6	3256	11	272	70
7	3424	16	259	73
8	3621	16	271	75
9	3811	13	260	78
10	4014	17	245	81
11	4229	13	257	77
12	3994	19	242	80
13	4179	15	253	83
14	4386	15	264	78
15	4208	13	249	82
16	4016	14	236	77
17	3835	16	226	81
18	4026	19	238	84
19	4258	16	251	88
20	4479	15	265	92
Iteration	(b) Profile B			
0	3000	15	300	60
1	4309	15	409	82
2	2635	7	268	43
3	3894	6	361	27
4	2241	8	197	39
5	1265	12	287	52
6	1699	16	420	75
7	1138	11	282	50
8	1543	12	153	67
9	2205	17	225	97
10	3198	10	332	51
11	4678	18	477	73
12	2588	8	315	48
13	1358	16	211	67
14	1794	17	307	98
15	2605	11	163	61
16	3719	17	240	87
17	2478	9	332	53
18	1507	16	466	76
19	2081	8	243	50
20	3053	17	149	70

j assigned to subtask s_i, where $\text{alloc}(i) \leqslant p_{\text{opt}}$ for s_i. Typically, multiple subtasks will be assigned to some of the same processors in a processor group, and then executed in a non-preemptive manner based on an ordering that obeys the precedence constraints (data dependencies) specified in the application task DAG. The scheduling string is a topological sort of the DAG (i.e., a valid total ordering of the partially ordered DAG).

In the initial population generation step, a predefined number of chromosomes are randomly created as follows. A new matching string is obtained by randomly assigning each subtask to a processor group. Scheduling strings are generated by performing a topological sort on the DAG and then applying a mutation operator to create random valid scheduling strings (i.e., schedules that are also valid topological sorts). The initial allocation string for each chromosome is generated by randomly selecting a value from 1 to p_{opt} as the number of processors allocated for each subtask (for the processor group type specified in the matching string). The solution from the ECT algorithm is also encoded as a chromosome and may be included in the initial population as a "seed." Indeed, it is common in GA applications to incorporate solutions from some non-evolutionary heuristic into the initial population, which may reduce the time needed for finding a satisfactory solution. In the GA used, it is guaranteed that the chromosomes in the initial population are distinct from each other. After the initial population is generated, the genetic operators crossover, mutation, and selection are applied one after the other for a number of generations (1000 in this study) or until the solution quality does not improve for 150 generations. Elitism is used to ensure that the best solution for each generation is explicitly protected from being modified by the mutation and crossover operations. The GA is executed ten times for a given dynamic parameter vector. To enhance diversity, only five of the ten runs include a chromosome generated by the ECT algorithm in the initial population. For more details of the GA, the reader is referred to [58].

5. Performance results

5.1. Introduction

Four approaches were compared in the experiments: (i) the On–Off approach; (ii) the ECT algorithm as a dynamic scheduling algorithm; (iii) the infeasible approach of using the GA as a real-time dynamic scheduling algorithm (referred to as GA on-line); and (iv) an ideal but impossible approach which uses the GA on-line with the exact (as yet unknown) dynamic parameters for the iteration to be executed (referred to as Ideal). The latter two schemes are included only to provide points of references for comparing the solution quality of the first two approaches.

5.2. Workload

To investigate the performance of the On–Off approach with the proposed dynamic parameter space partitioning and sampling methods, task graphs with four different structures were used. These graphs included in-tree graphs, out-tree graphs, fork-join graphs, and randomly structured graphs (see Fig. 5 for examples). Graphs with sizes 10, 50, 100, and 200 nodes were considered. For each graph structure and size, ten graphs were used in the simulation studies. Thus, a total of $4 \times 4 \times 10 = 160$ different graphs were

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 311 of 411

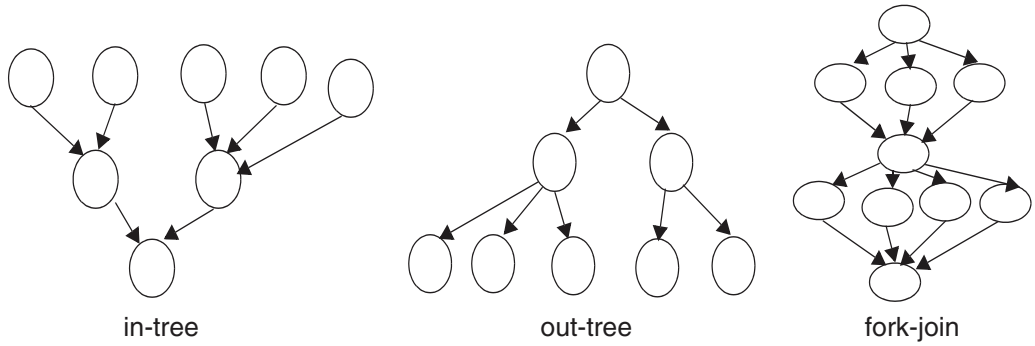


Fig. 5. Different common regular graph structures.

generated. According to [33], parallel algorithms can be classified based on the structure of their task graphs, and experience shows that most parallel algorithms belong to one of only a small number of classes. Examples of classes of task graphs are those representing asynchronous divide-and-conquer, multilevel or multiphase series-parallel, and pipelined parallel algorithms. Thus, in this study, tree structured task graphs are used to model many divide-and-conquer type of algorithms and fork-join task graphs model many multilevel or multiphase series-parallel algorithms. Randomly structured graphs were also included to represent parallel algorithms that do not fall into the other categories.

The randomly structured task graphs were generated as follows. Given the number of nodes n, the height (the number of levels) of the graph was randomly generated from a uniform distribution with range $[1 - 2\sqrt{n}]$. At each level, the number of nodes was randomly generated also using a uniform distribution but with the maximum value modified to guarantee that the final total number of nodes is n. The nodes at each level were then randomly connected to nodes at a higher level. The number of children for each node was chosen from a uniform distribution with range [0–7], but not exceeding the number of nodes specified for the next level. The regular graphs were generated according to their predefined structures.

In each graph, the coefficients of the subtask execution time equation (a_i, b_i, c_i) and inter-subtask communication time equation (d_{ij} and e_{ij}) were randomly generated from uniform distributions with ranges [10–100] and [1–10], respectively. The heterogeneity factors (h_{iu}'s) of these graphs were also randomly selected from a uniform distribution with range [0.5–20].

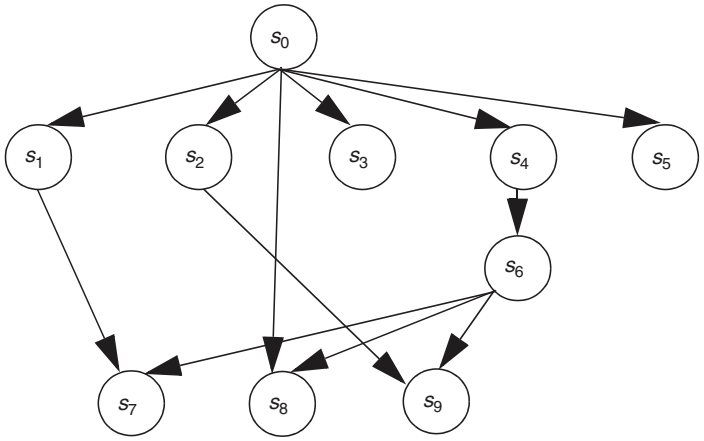
The heterogeneous platform shown in Fig. 3 was used throughout the experiments. Below are the parameters used in the experiments, unless otherwise stated.

- Ranges of the dynamic parameters: α: [1000–5000], β: [5–25], γ: [100–500], and μ: [20–100].
- Partitioning of the dynamic parameter space: the range of each dynamic parameter is partitioned into four equal intervals (i.e., $K = 4$, see Section 4.2) and, therefore, the mapping table stores $4^4 = 256$ mappings (i.e., there are 256 regions in the four dimensional space).
- Number of randomly chosen sample scenarios within each partition (hyper-rectangle) of the dynamic parameter space: 10 (i.e., $N = 10$, see Section 4.2).
- The GA is executed 10 times for each sample scenario, from which the best mapping is chosen (thus, for a single graph, the GA was executed a total of $K^4 \times N \times 10 = 256 \times 10 \times 10 = 25,600$ times to build the mapping table).
- Estimated reconfiguration time: 1000.
- Crossover and mutation probabilities for the GA: both 0.4 (these values were chosen according to the light/moderate load results in [58]).

Two randomly generated 20-iteration execution profiles of dynamic parameters were used for each graph (see Table 2). The dynamic parameters for Profile B change eight times more rapidly, on average, than those for Profile A. In the profiles, iteration 0 is the initialization iteration. In this study, the dynamic parameter values of iteration 0 are chosen to be the mean values of the respective dynamic parameter ranges. The dynamic parameter values shown on iteration i ($i > 0$) simulate the actual dynamic parameter values *observed* after the task graph finishes iteration i execution.

Both Profiles A and B were generated randomly based on a single parameter: the mean percentage change in dynamic parameter values, called Δ. Specifically, given Δ, an increment factor, denoted by δ_{i-1}, was randomly chosen from a uniform distribution with a range [0.5Δ–1.5Δ], and its sign had a 0.5 probability of being positive. Then, a dynamic parameter for iteration i, say μ_i, was given by: $\mu_i = \mu_{i-1} \pm \delta_{i-1}\mu_{i-1}$ (the same is done for α and γ; β is given by μ/ζ where ζ was randomly chosen from a uniform distribution with a range of [4.0, 6.0]). In generating the profiles, the dynamic parameters are bounded within the specified ranges using the following method. If, say, a dynamic parameter goes beyond the upper bound (because of a positive δ_{i-1}), then the value of δ_{i-1} is negated to force the dynamic parameter to decrease rather than increase, and the reverse

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 312 of 411



(a)

s_i	a_i	b_i	c_i
s_1	9	24	49
s_2	42	61	9
s_3	42	50	43
s_4	2	34	47
s_5	9	10	38
s_8	33	59	76
s_7	63	45	29
s_8	56	14	54
s_9	48	68	10
	36	25	29

(b)

$s_i \rightarrow s_j$	d_{ij}	e_{ij}
$s_0 \rightarrow s_1$	5	3
$s_0 \rightarrow s_2$	6	6
$s_0 \rightarrow s_3$	2	1
$s_0 \rightarrow s_4$	7	5
$s_0 \rightarrow s_5$	3	7
$s_0 \rightarrow s_8$	5	6
$s_1 \rightarrow s_7$	3	2
$s_2 \rightarrow s_9$	7	7
$s_4 \rightarrow s_6$	4	2
$s_6 \rightarrow s_7$	5	6
$s_6 \rightarrow s_8$	7	8
$s_6 \rightarrow s_9$	9	10

(c)

s_i	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_1	0.4897	0.6815	0.7711	0.7503
s_2	0.2828	0.7129	0.4511	0.2725
s_3	0.2575	0.8511	0.5096	0.8779
s_4	0.6337	0.6921	0.8479	0.3451
s_5	0.8283	0.2745	0.4114	0.2836
s_8	0.7267	0.3124	0.2600	0.3354
s_7	0.3932	0.7026	0.8072	0.8066
s_8	0.5276	0.7990	0.5081	0.7942
s_9	0.5876	0.3863	0.6515	0.6472
	0.8760	0.8794	0.6965	0.2407

(d)

Fig. 6. (a) A ten-node randomly generated task graph; (b) coefficients of the subtask execution time equations; (c) coefficients of the inter-subtask communication data equations; (d) heterogeneity factors h_{iu} for the subtask.

is done when the dynamic parameter goes below the lower bound. Thus, the extent of change in dynamic parameter values could be controlled by specifying Δ: a higher value of Δ implies a more swiftly changing execution profile, which, in turn, can model a series of swiftly changing scenes in an input stream of images. The values of Δ used for Profiles A and B are 5% and 40%,respectively.

5.3. An illustrative example

To examine the performance of the On–Off approach, first consider the results of scheduling a 10-node random task graph using the two execution profiles. The structure and parameters of the 10-node random task graph are shown in Fig. 6. Detailed results of using the four approaches for Profile A are shown in Table 3. Below are the definitions of the data columns.

- $t(map[i-1])$: this is the task execution time of iteration i using the mapping *chosen* at the end of iteration $i-1$, denoted by map$[i-1]$. Here, it should be noted that at the end of iteration $i-1$, a new mapping will be determined but such a mapping would not be used for iteration i if the reconfiguration time offsets the gain of remapping. Thus, a mapping chosen at the end of iteration $i-1$ could be a new mapping or the same mapping used for iteration $i-1$. In the case of On–Off, the new mapping considered is the representative mapping for the region containing the dynamic parameter values at iteration $i-1$ stored in the mapping table. In the case of ECT, the new mapping considered is the one determined using the ECT

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 313 of 411

Table 3
Results for the ten-node random graph using Profile A ("total" below is the total task execution time for 20 iterations)

i	On–Off			ECT			GA On-line	Ideal
	t(map[$i-1$])	t(tab[$i-1$])	rc	t(map[$i-1$])	t(ect[$i-1$])	rc	t(ga[$i-1$])	t(ga[i])
0	—	54,391	1000	—	74,516	1000	—	46,228
1	47,508	46,045	1000	76,384	76,029	0	47,508	47,508
2	47,993	47,286	0	79,227	75,866	1000	49,095	45,280
3	48,301	51,679	0	75,375	74,586	0	50,306	45,856
4	48,921	51,679	0	76,912	74,755	1000	48,520	43,611
5	44,035	51,679	0	74,966	71,491	1000	53,579	40,937
6	49,121	51,679	0	76,226	75,020	1000	48,410	46,496
7	53,225	52,891	0	79,468	77,336	1000	51,161	47,580
8	55,303	52,891	1000	81,944	79,878	1000	55,447	52,809
9	56,355	51,679	1000	83,053	82,556	0	52,846	50,539
10	63,288	63,958	0	86,042	87,381	0	55,887	53,733
11	56,631	52,610	1000	82,874	88,226	0	58,984	51,973
12	58,025	55,142	1000	87,122	81,860	1000	63,183	53,733
13	57,410	63,958	0	87,644	87,182	0	57,948	54,311
14	59,774	53,073	1000	85,449	87,898	0	56,267	52,328
15	58,932	60,754	0	86,152	87,311	0	59,472	55,476
16	55,878	58,723	0	81,710	81,883	0	55,878	55,878
17	58,200	59,774	0	81,434	76,982	1000	56,981	51,227
18	60,966	63,958	0	88,524	81,581	1000	56,961	54,483
19	63,071	63,958	0	91,701	84,474	1000	60,043	59,846
20	65,608	—	—	95,498	93,014	—	58,528	53,842
Total	1,115,545			1,688,705			1,097,004	1,065,040

algorithm with the exact dynamic parameters at iteration $i-1$.

- $t(tab[i-1])$: this is the task execution time of the representative mapping stored in the off-line mapping table, denoted by tab[$i-1$], for the region containing the dynamic parameter values at iteration $i-1$ stored in the mapping table. This is the value stored in the mapping table, instead of the task execution time by applying tab[$i-1$] to the dynamic parameters at iteration $i-1$. The mapping tab[$i-1$] may or may not be chosen for iteration i.
- rc: the reconfiguration cost, if remapping is performed.
- $t(ect[i-1])$: this is the execution time of the mapping, denoted by ect[$i-1$], determined using the ECT algorithm with the dynamic parameters at iteration $i-1$. The mapping may or may not be chosen for iteration i.
- $t(ga[i-1])$: this is the task execution time of iteration i by applying the mapping determined by the GA using the dynamic parameter values from iteration $i-1$. This mapping is applied whether the remapping is justified by the gain or not. The mapping found at iteration $i-1$ (by the GA using the dynamic parameter values from iteration $i-1$) is incorporated in the initial population of the GA at iteration i. This GA on-line method is included for comparison only; applying the GA on-line for dynamic scheduling is infeasible due to the long execution time required by the GA. The GA on-line approach may require more reconfigurations than the On–Off method because it searches for a customized mapping at each iteration. Thus, to conservatively compare these approaches, the reconfiguration time for GA on-line is not considered.
- $t(ga[i])$: this is the task execution time of iteration i determined by the GA using the exact dynamic parameter values from iteration i. This is, therefore, the ideal case, and is impossible in practice because the actual values of the dynamic parameters for iteration i cannot be known before the execution of iteration i begins. Furthermore, the solutions found by both the On–Off (tab[$i-1$]) and the GA on-line (ga[$i-1$]) approaches are incorporated into the initial population of the GA. This is done to determine the "best" possible solution as a reference for comparison. For reasons similar to those given for GA on-line, reconfiguration time for GA Ideal is not considered.

As can be seen from Table 3, the On–Off approach of dynamically using off-line-derived mappings generated much smaller total execution time (1, 115, 545) compared to that of using the ECT algorithm (1, 668, 705). The improvement is approximately 33%. The On–Off approach consistently resulted in performance that was comparable to the infeasible GA on-line scheme (about 2% worse) and was only marginally outperformed by the Ideal (but impossible) method (about 5% worse). Recall that the strategy underlying the On–Off approach is to provide a mechanism by which mappings derived from time-consuming static heuristics, such as the GA, can be employed on-line; i.e., using the On–Off approach allows the power of the GA heuristic to be

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 314 of 411

achieved on-line. Indeed, one very interesting observation is that at some iterations (i.e., iterations 2, 3, 5, 8, 11, 12, 13, and 15), the On–Off approach generated shorter mapping execution times than the GA on-line (e.g., see Table 4 for the substantially different mappings used by the four approaches at iteration 5). This can be explained by the fact that the GA on-line mapping is optimized for the $i-1$ iterations dynamic parameters and thus, this mapping may not be very suitable for the ith iteration if the parameters vary considerably. The On–Off approach, however, is more robust to rapidly changing dynamic parameters because its mapping is based on good average performance over a wide range of dynamic parameters.

The results of scheduling the ten-node random graphs for Profile B are shown in Table 5. As expected, more reconfigurations were performed for this profile than for Profile A because the dynamic parameter values change much faster. Once again the ECT approach resulted in the worst performance with a total execution time that was approximately 45% longer than that of the On–Off approach. The On–Off approach was only slightly outperformed by the infeasible GA on-line method (about 5%) and about 20% worse than the impossible Ideal method. It should be noted that the higher degradation is partially due to the higher reconfiguration overhead of the On–Off method, which is not included in the GA on-line or Ideal execution times. Similar to the case of Profile A, at some iterations (i.e., iterations 1, 2, 8, 9, 12, and 17) the On–Off approach produced shorter task execution times than those of the infeasible GA on-line scheme.

5.4. Effectiveness of the sampling strategy

Some additional experiments using larger graphs were conducted to further test the effectiveness of the sampling strategy used. Specifically, ten 50-node random graphs were used and the following different approaches to generate representative mappings were compared:

- the mid-point of the hyper-rectangle is chosen as the only sample scenario for generating the representative mapping (called Scheme 1),
- a randomly selected point within the hyper-rectangle is chosen as the only sample scenario (called Scheme 2);
- ten sample scenarios in the hyper-rectangle are examined but for each sample scenario the GA is executed without incorporating the solution generated by the ECT algorithm as one of the members in the initial population (and the mappings of the ten sample scenarios are applied to all other sample scenarios in the hyper-rectangle, with the mapping giving the best average performance selected) (called Scheme 3);
- the approach used throughout all previous experiments—like Scheme 3 except the GA is executed with the ECT solution as a seed chromosomes (called Scheme 4).

Table 4
Mappings for iteration 5 of Profile A used by the (a) On–Off approach (adopted after iteration 1), (b) ECT approach (adopted after iteration 4), (c) GA on-line approach, and (d) Ideal approach

s_i	Proc type	ρ	$t_u(s_i)$	Comm.	C_{uv}
(a) *On–Off approach*					
s_0	1	10	11,217	$s_0 \to s_1$	77
s_1	1	13	10,864	$s_0 \to s_2$	184
s_2	2	10	13,367	$s_0 \to s_3$	34
s_3	2	5	12,199	$s_0 \to s_4$	2551
s_4	3	6	4190	$s_0 \to s_5$	212
s_5	2	2	18,553	$s_0 \to s_8$	1523
s_6	0	9	12,175	$s_1 \to s_7$	64
s_7	2	8	18,346	$s_2 \to s_9$	3048
s_8	0	15	9355	$s_4 \to s_6$	1562
s_9	3	10	4738	$s_6 \to s_7$	2281
				$s_6 \to s_8$	223
				$s_6 \to s_9$	7600
(b) *ECT approach*					
s_0	0	10	8060	$s_0 \to s_1$	2321
s_1	3	4	9906	$s_0 \to s_2$	1526
s_2	1	16	18,558	$s_0 \to s_3$	792
s_3	3	16	4928	$s_0 \to s_4$	3839
s_4	3	4	4825	$s_0 \to s_5$	194
s_5	0	14	21,664	$s_0 \to s_8$	2281
s_6	0	2	41,444	$s_1 \to s_7$	1551
s_7	0	16	13,440	$s_2 \to s_9$	176
s_8	2	12	11,798	$s_4 \to s_6$	1562
s_9	1	14	14,655	$s_6 \to s_7$	167
				$s_6 \to s_8$	3040
				$s_6 \to s_9$	2534
(c) *GA on-line approach*					
s_0	1	10	11,217	$s_0 \to s_1$	95
s_1	2	10	8090	$s_0 \to s_2$	1526
s_2	0	7	8100	$s_0 \to s_3$	26
s_3	1	2	10,836	$s_0 \to s_4$	2551
s_4	3	10	3691	$s_0 \to s_5$	1764
s_5	0	10	23,510	$s_0 \to s_8$	1523
s_6	2	8	27,098	$s_1 \to s_7$	64
s_7	1	11	24,203	$s_2 \to s_9$	5334
s_8	0	15	9355	$s_4 \to s_6$	891
s_9	3	16	3787	$s_6 \to s_7$	183
				$s_6 \to s_8$	3040
				$s_6 \to s_9$	4343
(d) *Ideal approach*					
s_0	0	10	8060	$s_0 \to s_1$	85
s_1	0	5	8517	$s_0 \to s_2$	2287
s_2	2	11	12,811	$s_0 \to s_3$	29
s_3	0	1	11,600	$s_0 \to s_4$	3839
s_4	3	14	3484	$s_0 \to s_5$	1764
s_5	1	4	14,568	$s_0 \to s_8$	1523
s_6	1	16	15,449	$s_1 \to s_7$	57
s_7	0	10	16,799	$s_2 \to s_9$	3048
s_8	1	8	9183	$s_4 \to s_6$	1033
s_9	3	14	4011	$s_6 \to s_7$	1523
				$s_6 \to s_8$	201
				$s_6 \to s_9$	5058

Note that Scheme 1 is equivalent to using a uniform distribution for representative values of dynamic parameters suggested in [10] (recall that the only other approach in [10] forced the application developer to guess what representative values to use). The above four approaches were applied to ten 50-node random graphs using Profile A and the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 315 of 411

Table 5
Results for the ten-node random graph using Profile B ("total" below is the task execution time for 20 iterations)

i	On–Off			ECT			GA On-line	Ideal
	$t(\text{map}[i-1])$	$t(\text{tab}[i-1])$	rc	$t(\text{map}[i-1])$	$t(\text{ect}[i-1])$	rc	$t(\text{ga}[i-1])$	$t(\text{ga}[i])$
0	—	54,391	1000	—	76,029	1000	—	47,980
1	61,988	69,217	0	107,754	103,630	1000	66,150	61,988
2	36,411	38,707	0	63,998	61,852	1000	45,692	34,972
3	49,142	45,126	1000	72,559	79,625	0	44,712	40,318
4	43,516	26,908	1000	51,686	51,648	0	38,337	28,418
5	38,740	34,726	1000	64,508	49,440	1000	35,719	32,825
6	50,402	48,248	1000	74,503	74,365	0	44,250	29,231
7	34,390	50,402	0	50,107	49,296	0	29,842	27,753
8	41,947	33,684	1000	62,863	45,772	1000	48,713	31,601
9	45,940	48,911	0	83,088	85,888	0	48,095	44,184
10	57,662	48,178	1000	75,503	76,404	0	55,606	42,139
11	64,180	62,401	1000	111,327	112,449	0	69,058	61,897
12	39,930	42,374	0	67,829	68,907	0	40,476	39,930
13	46,949	40,904	1000	59,054	47,534	1000	46,350	31,203
14	46,677	45,483	1000	84,529	67,120	1000	45,041	41,910
15	58,622	41,498	1000	54,991	56,205	0	34,473	26,400
16	61,211	55,142	1000	78,358	80,091	0	57,254	51,032
17	46,193	42,374	1000	69,973	72,405	0	46,243	36,274
18	56,042	41,947	1000	85,659	72,229	1000	55,192	44,846
19	49,145	38,707	1000	52,644	57,459	0	47,570	31,678
20	51,900	—	—	68,733	63,337	—	48,931	43,881
Total	995,987			1,447,666			947,704	830,460

total mapping execution times were noted. The averages of these execution times were determined and the results were normalized with respect to those of Scheme 4. The results obtained are as follows: 1.23 (Scheme 1), 1.19 (Scheme 2), 1.16 (Scheme 3), and 1.00 (Scheme 4). These results lead to the conclusion that the relationship between the parameters space and the mappings space is highly irregular and, as such, more random sample scenarios are needed to more accurately "characterize" a good representative mapping for a hyper-rectangle. Finally, as expected, the solutions of the GA without using mappings determined by ECT are worse. Given these findings, Scheme 4 was used throughout all other experiments.

5.5. Results for larger graphs with different structures

To investigate the effects of graph sizes and structures on the performance of the On–Off approach, more extensive experiments were done for larger task graphs (recall that for each type of graphs, the test set contains ten graphs for each size of 10, 50, 100, and 200 nodes). Fig. 7 shows the average normalized total execution times of the ECT, On–Off, and GA on-line approaches with respect to the Ideal method. The normalized total execution time of each test case is calculated by dividing the total execution time of a particular approach (e.g., ECT) by that of the Ideal method. Each point on the curves gives the average value of ten test cases. Recall that the GA on-line approach is infeasible, the Ideal approach is impossible, and neither include reconfiguration time in their resultant execution time (the ECT and On–Off methods, however, do include reconfiguration time).

The execution time of the infeasible GA on-line was at most 20% greater than the Ideal approach (which assumed advance knowledge of the iteration i parameter values). The execution time of the On–Off was at most 30% greater than the *infeasible* GA on-line. Moreover, comparing the results of Profile A with that of Profile B, in most cases there was no significant increase in relative performance degradation for increasing incremental changes (Δ) in dynamic parameter values.

However, the ECT approach performed much worse, particularly for large random graphs. An explanation for this phenomenon is that because the ECT algorithm employs a strictly greedy scheduling method, the effect of making mistakes at early stages of scheduling can propagate until the whole graph is completely scheduled. The adverse impact of such a greedy approach can be more profound for larger graphs. The ECT approach did not perform as poorly for the tree and fork-join graphs because these graphs tend to contain long chains of subtasks that the ECT algorithm tends to handle well [57]. Fig. 8 shows the 95% confidence intervals (i.e., given the calculated mean over the 40 test cases, the probability that the true mean is in the interval shown is 0.95; see [12]) corresponding to the results shown in Fig. 7

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 316 of 411

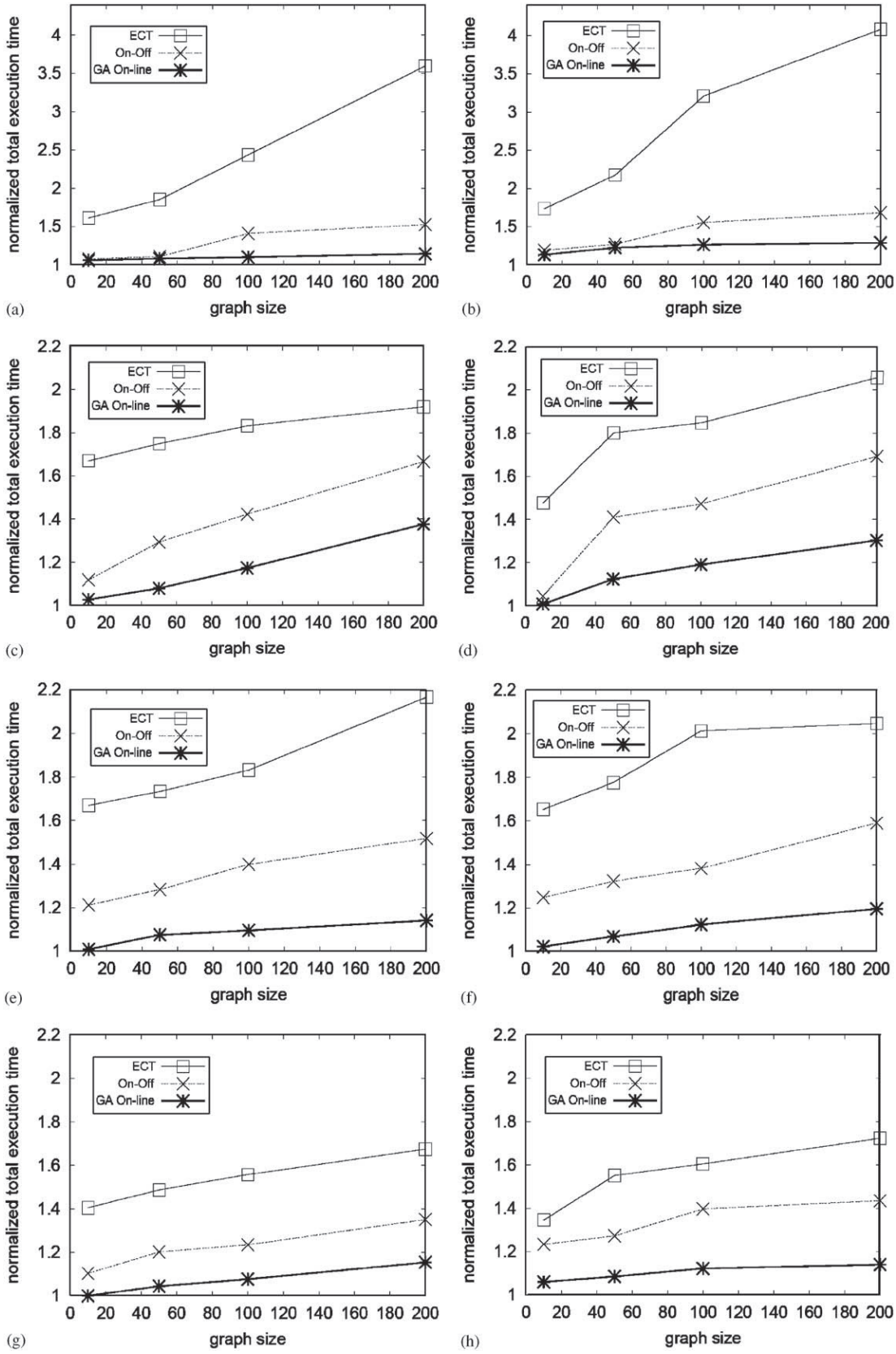


Fig. 7. Average normalized total execution times for the four types of task graphs: (a) random graphs, Profile A, (b) random graphs, Profile B, (c) in-tree graphs, Profile A, (d) in-tree graphs, Profile B, (e) out-tree graphs, Profile A, (f) out-tree graphs, Profile B, (g) fork-join graphs, Profile A and (h) fork-join graphs, Profile B (note that the scale for graphs (a) and (b) is twice that of the other graphs.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 317 of 411

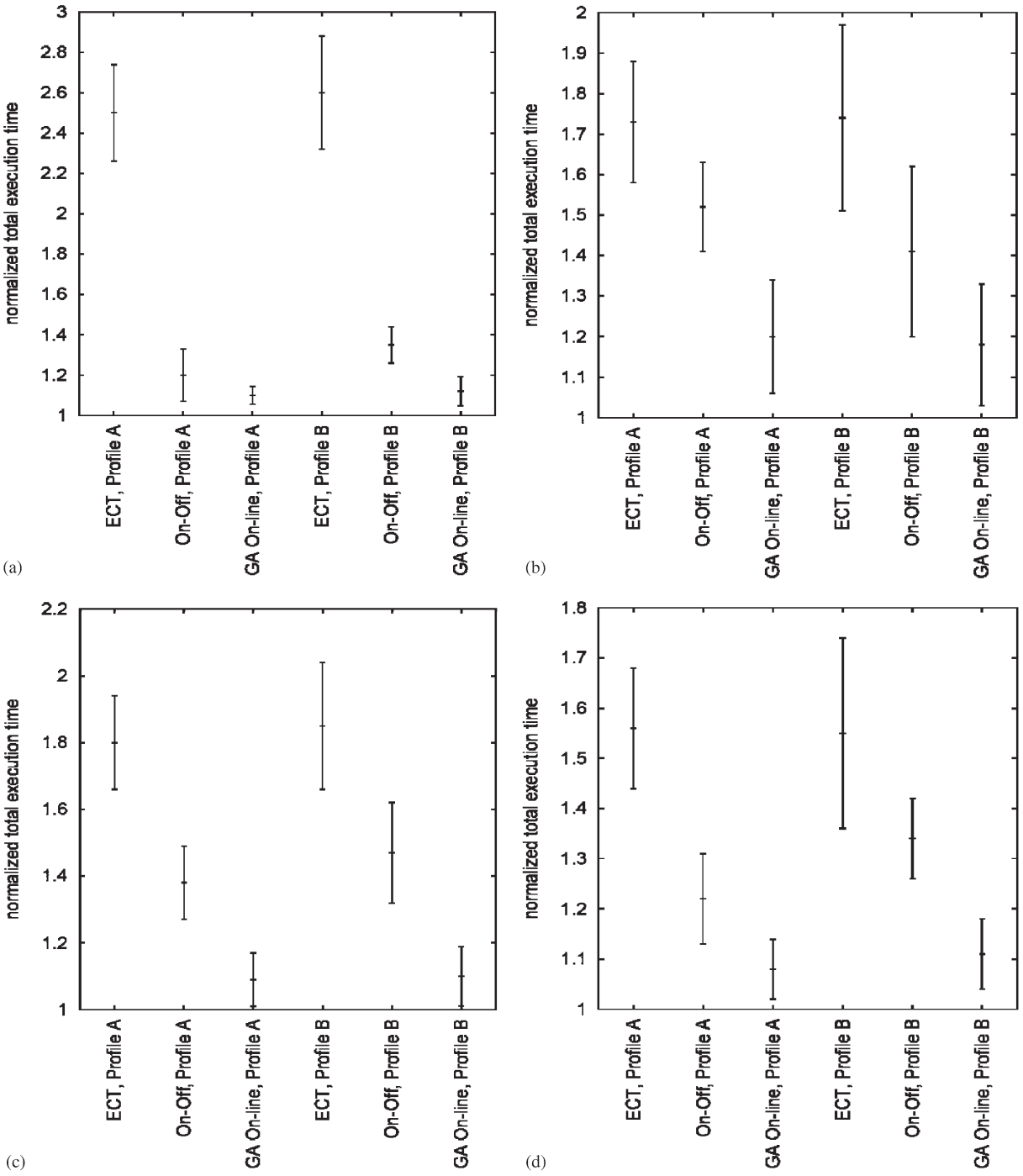


Fig. 8. The 95% confidence intervals of the normalized total execution times produced by the three approaches for all the four types of task graphs: (a) random graphs, (b) in-tree graphs, (c) out-tree graphs and (d) fork-join graphs.

and the absolute total execution times of the Ideal method are shown in Fig. 9.

5.6. Effect of the reconfiguration cost

An experiment was also conducted to explore the effect of increasing the reconfiguration cost on the performance of the On–Off approach. Ten 100-node random task graphs and Profile B (with 20 iterations) were used. The reconfiguration cost was varied as: 1000, 5000, 25,000, 50,000, and 100,000, which approximately correspond to 0.2%, 1%, 5%, 10%, and 20% of the execution time of a single iteration. The average total mapping execution times are shown in Table 6. As can be seen, the total execution times of the ECT and On–Off approaches increase moderately despite the fact that the reconfiguration cost varied over a wide range. A scrutiny

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 318 of 411

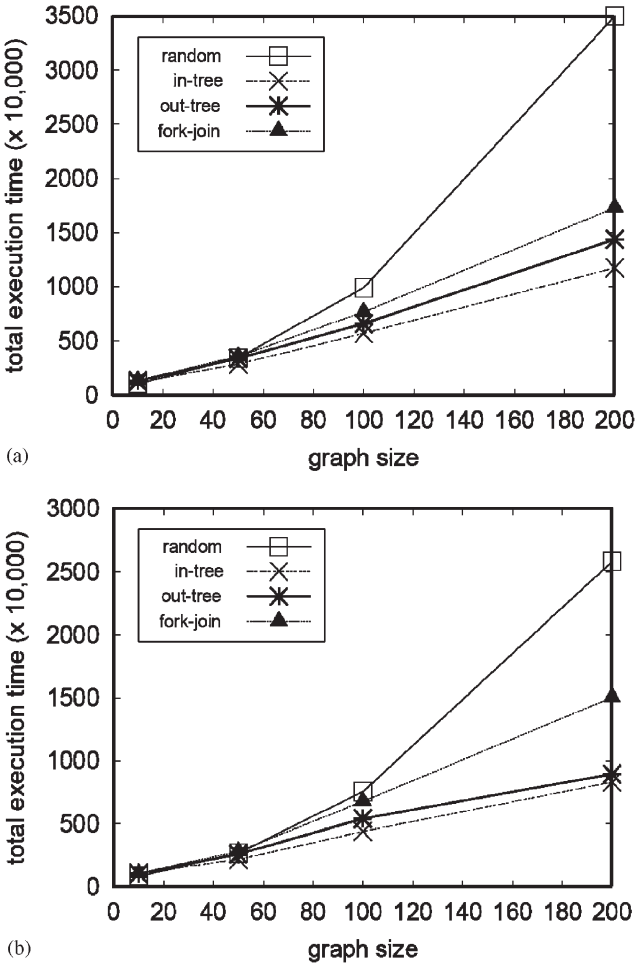


Fig. 9. Average total execution times of the Ideal approach for the four types of task graphs: (a) Profile A and (b) Profile B.

of the detailed execution traces revealed that for the ECT and On–Off approaches, a higher reconfiguration cost simply reduces the number of switches to a new mapping from one iteration to another. If reconfiguration cost is small, more remappings are performed but the aggregate reconfiguration costs do not considerably affect the total time. It appears that increasing the reconfiguration cost does not critically affect the total execution time. Most importantly, even though the total execution times of On–Off increase as expected, the On–Off approach still performs very well with respect to the other approaches.

5.7. Comparison with using a single mapping

It is interesting to compare the performance of all approaches considered thus far with a *static* approach that works by find the "best" mapping in the On–Off table. Specifically, each mapping from the 256 regions is applied to all of the 256 dynamic parameter vectors. The mapping that gives the shortest average execution time (across the 256 scenarios) is selected as the static mapping and is used in all the iterations in an execution profile. In this experiment, a suite of forty 200-node graphs was used: for each of the graph structures (random, in-tree, out-tree, and fork-join), ten graphs were used. The mapping approaches were applied to the task graphs using ten different 200-iteration execution profiles with average percentage change Δ varied from 5% to 50% in increments of 5%. These results are shown in Fig. 10. As can be seen, the relative performance of the On–Off, GA on-line, and ECT approaches is quite consistent. Furthermore, the On–Off method outperformed the static approach considerably. On the other hand, it is interesting to see that the static approach gave better performance than the ECT algorithm for test cases with small changes in dynamic parameter values, indicating that the solution quality of the GA is indeed much better than the ECT heuristic. This is due to the fact that, for this experiment, all execution profiles started in the middle of the dynamic parameter space for which the best average mapping was quite good. However, as Δ increased, the dynamic parameter values strayed further from the middle and the static approach generated progressively worse solutions (although still not as bad as a typical static mapping). For large values of Δ (e.g., larger than 25%), even the ECT approach outperformed the static approach. This illustrates that using the same mapping is not desirable for scheduling a task graph with vigorously changing attributes. In contrast, the On–Off approach was much less significantly affected by quickly varying dynamic parameters, only increasing from 1.213 to 1.297 while Δ increased from 5% to 50%.

5.8. Running time of the mapping algorithms

A limitation of the On–Off approach is the relatively long running time needed to build the mapping table. For example, while the ECT algorithm took 0.5 s to generate a mapping for a 100-node graph, the GA used in the On–Off approach required about 10 min off-line for each mapping table entry. However, because the mapping table is to be built off-line and the target heterogeneous task graph is to be used as a production job, some extra time is affordable. In addition, the efficiency of the On–Off approach can be further improved because it is possible to parallelize the GA used.

6. Conclusions

A distinctive feature of the semi-static strategy is that it approaches the off-line mapping quality of a genetic algorithm (GA) with on-line efficiency. To make the semi-static mapping methodology complete, a new technique was designed for selecting off-line mappings through partitioning and sampling the dynamic parameter space of the heterogeneous application. Experimental results indicate that with the proposed method for determining the representative static mappings, the semi-static approach is effective in that it consistently outperformed a fast dynamic mapping heuristic,

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 319 of 411

Table 6
The total execution times (average over ten graphs for each reconfiguration cost) for 100-node random task graphs using (a) Profile A, and (b) Profile B (20 iterations). The number of reconfigurations of ECT and On–Off are shown in the parentheses

Reconfig. cost	ECT	On–Off	GA on-line	Ideal
(a) *Profile A*				
1000	25,749,859	10,581,274	9,219,447	8,025,182
	(13)	(10)	—	—
5000	26,108,420	10,809,310	9,219,447	8,025,182
	(11)	(8)	—	—
25,000	26,590,023	11,093,256	9,219,447	8,025,182
	(8)	(6)	—	—
50,000	27,092,528	11,321,890	9,219,447	8,025,182
	(7)	(6)	—	—
100,000	27,502,753	11,740,731	9,219,447	8,025,182
	(7)	(6)	—	—
(a) *Profile B*				
1000	27,259,122	11,124,981	10,954,377	9,014,579
	(14)	(12)	—	—
5000	27,850,353	11,370,890	10,954,377	9,014,579
	(13)	(10)	—	—
25,000	28,601,246	11,691,931	10,954,377	9,014,579
	(11)	(8)	—	—
50,000	29,381,251	12,017,445	10,954,377	9,014,579
	(9)	(7)	—	—
100,000	29,991,021	12,529,694	10,954,377	9,014,579
	(9)	(7)	—	—

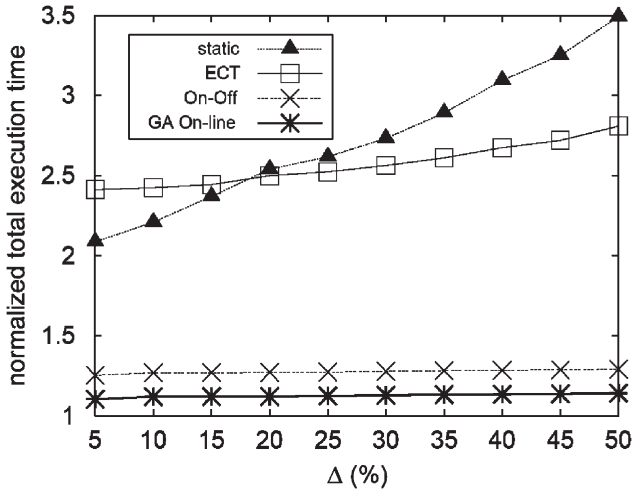


Fig. 10. Average total execution times (normalized by the execution times of the Ideal approach) for a suite of forty 200-node task graphs using various values of Δ (200-iteration profiles were used).

and yields reasonable performance compared with the infeasible approach of directly using the GA on-line for a wide range of task graph structures. Indeed, the sampling strategy performs even better than using the GA on-line (based on parameter values of the previous iteration) in several instances. Furthermore, it is found that the performance of the methodology is effective even for handling swiftly changing dynamic parameters in these heterogeneous applications. In addition, the semi-static approach outperformed the best static method by a considerable margin. A limitation of such a semi-static approach is the additional off-line execution time needed to build the mapping table. However, because the mapping table is built off-line and the target heterogeneous task graph is used as a production job, some extra time is affordable.

Acknowledgments

The authors thank the anonymous reviewers (in particular, Reviewer 1) for the constructive comments that have helped to improve this paper. Thanks are also due to Muthucumaru Maheswaran and Tracy D. Braun for their helpful suggestions. A preliminary version of portions of this paper appeared in the Proceedings of the Fourth International Symposium on Parallel Architectures, Algorithms, and Networks.

References

[1] I. Ahmad, A. Ghafoor, Semi-distributed load balancing for massively parallel multicomputer systems, IEEE Trans. Software Eng. 17 (10) (October 1991) 987–1004.

[2] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, Characterizing resource allocation heuristics for heterogeneous computing systems, in: A.R. Hurson (Ed.), Advances in Computers, Parallel, Distributed, and Pervasive Computing, vol. 63, Elsevier, Amsterdam, The Netherlands, 2005, pp. 91–128.

[3] H.M. Alnuweiri, V.K. Prasanna, Parallel architectures and algorithms for image component labeling, IEEE Trans. Pattern Anal. Mach. Intell. 14 (10) (October 1992) 1014–1034.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 320 of 411

[4] P. Baglietto, M. Maresca, M. Migliardi, N. Zingirian, Image processing on high-performance RISC systems, Proc. IEEE 84 (7) (July 1996) 917–930.
[5] R. Bajaj, D.P. Agrawal, Improving scheduling of tasks in a heterogeneous environment, IEEE Trans. Parallel Distributed Systems 15 (2) (February 2004) 107–118.
[6] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, Y. Robert, Scheduling strategies for master–slave tasking on heterogeneous processor platforms, IEEE Trans. Parallel Distributed Systems 15 (4) (April 2004) 319–300.
[7] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, R.F. Freund, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distributed Comput. 61 (6) (June 2001) 810–837.
[8] L.L. Bölöni, D.C. Marinescu, Robust scheduling of metaprograms, J. Scheduling 5 (5) (September 2002) 395–412.
[9] J.R. Budenske, R.S. Ramanujan, H.J. Siegel, Modeling ATR applications for intelligent execution upon a heterogeneous computing platform, Proceedings of the 1997 IEEE International Conference on Systems, Man, and Cybernetics, October 1997, pp. 649–656.
[10] J.R. Budenske, R.S. Ramanujam, H.J. Siegel, A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms, J. Supercomput. 12 (4) (October 1998) 387–406.
[11] E.A. Carmona, M.D. Rice, Modeling the serial and parallel fractions of a parallel program, J. Parallel Distributed Comput. 13 (3) (November 1991) 286–298.
[12] C.G. Cassandras, Discrete Event Systems: Modeling and Performance Analysis, Irwin, Homewood, IL, 1993.
[13] A. Choudhary, R. Thakur, Connected component labeling on coarse grain parallel computers: an experimental study, J. Parallel Distributed Comput. 20 (1) (January 1994) 78–83.
[14] C.H. Chu, E.J. Delp, H.J. Siegel, F.J. Weil, A.B. Whinston, A model for an intelligent operating system for executing image understanding tasks on a reconfigurable parallel architectures, J. Parallel Distributed Comput. 6 (3) (June 1989) 598–622.
[15] N. Copty, S. Ranka, G. Fox, R.V. Shankar, A data parallel algorithm for solving the region growing problem on the connection machine, J. Parallel Distributed Comput. 21 (1) (April 1994) 160–168.
[16] P. David, S. Balakirsky, D. Hillis, A real-time automatic target acquisition system, Proceedings of the International Conference on Unmanned Vehicle Systems, July 1990, pp.183–198.
[17] P. David, P. Emmerman, S. Ho, A scalable architecture system for automatic target recognition, Proceedings of the 13th AIAA/IEEE Digital Avionics Systems Conference, October 1994, pp. 414–420.
[18] T.H. Einstein, Mercury Computer Systems' Modular Heterogeneous RACE Multicomputer, Proceedings of the Sixth Heterogeneous Computing Workshop, April 1997, pp. 60–71.
[19] M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996.
[20] M.M. Eshaghian, J.G. Nash, M.E. Shaaban, D.B. Shu, Heterogeneous algorithms for image understanding architecture, Proceedings of the Workshop on Computer Architectures for Machine Perception, December 1993, pp. 286–292.
[21] T. Fahringer, Estimating and optimizing performance for parallel programs, Computer 28 (11) (November 1995) 47–56.
[22] D. Fernandez-Baca, Allocating modules to processors in a distributed systems, IEEE Trans. Software Eng. SE-15 (11) (November 1989) 1427–1436.
[23] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
[24] A. Ghafoor, J. Yang, A distributed heterogeneous supercomputing management system, Computer 26 (6) (June 1993) 78–86.
[25] N. Guil, E.L. Zapata, Fast Hough transform on multiprocessors: a branch and bound approach, J. Parallel Distributed Comput. 45 (1) (August 1997) 82–89.
[26] K. Hwang, Z. Xu, M. Arakawa, Benchmark evaluation of the IBM SP2 for parallel signal processing, IEEE Trans. Parallel Distributed Systems 7 (5) (May 1996) 522–536.
[27] A.A. Khokhar, G.W. Cook, L.H. Jamieson, E.J. Delp, Coarse-grained algorithms and implementations of structural indexing-based object recognition on Intel touchstone delta, Proceedings of the 12th International Conference on Pattern Recognition, vol. 3, October 1994, pp. 279–283.
[28] A.A. Khokhar, V.K. Prasanna, M. Shaaban, C.-L. Wang, Heterogeneous computing: challenges and opportunities, Computer 26 (6) (June 1993) 18–27.
[29] Y.-K. Kwok, I. Ahmad, Dynamic critical-path scheduling: an effective technique for allocating task graphs onto multiprocessors, IEEE Trans. Parallel Distributed Systems 7 (5) (May 1996) 506–521.
[30] G.O. Ladd Jr., Practical issues in heterogeneous processing systems for military applications, Proceedings of the Sixth Heterogeneous Computing Workshop, April 1997, pp. 162–169.
[31] C. Lee, Y.-F. Wang, T. Yang, Global optimization for mapping parallel image processing tasks on distributed memory machines, J. Parallel Distributed Comput. 45 (1) (August 1997) 29–45.
[32] Q. Liu, R.J. Sclabassi, M.L. Scheuer, M. Sun, A two-step method for compression of medical monitoring video, Proceedings of the 25th Annual International IEEE Engineering in Medicine and Biology Society, vol. 1, September 2003, pp. 845–848.
[33] S. Madala, J.B. Sinclair, Performance of synchronous parallel algorithms with regular structures, IEEE Trans. Parallel Distributed Systems 2 (1) (January 1991) 105–116.
[34] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distributed Comput. 59 (2) (November 1999) 107–131.
[35] M. Maheswaran, T.D. Braun, H.J. Siegel, Heterogeneous distributed computing, in: J. Webster (Ed.), Encyclopedia of Electrical and Electronics Engineering, vol. 8, Wiley, New York, NY, 1999, pp. 679–690.
[36] D.C. Marinescu, J.R. Rice, On high level characterization of parallelism, J. Parallel Distributed Comput. 20 (1) (January 1994) 107–113.
[37] D.G. Morris III, D.K. Lowenthal, Accurate data redistribution cost estimation in software distributed shared memory systems, Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'2001), June 2001, pp. 62–71.
[38] D.M. Nicol, J.H. Saltz, Dynamic remapping of parallel computations with varying resource demands, IEEE Trans. Comput. 37 (9) (September 1988) 1073–1087.
[39] C.-I. Park, T.-Y. Choe, An optimal scheduling algorithm based on task duplication, IEEE Trans. Comput. 51 (4) (April 2002) 444–448.
[40] A. Radulescu, C. Nicolescu, A.J.C. van Gemund, P.P. Jonker, CPR: mixed task and data parallel scheduling for distributed systems, Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS'2001), April 2001.
[41] S. Ramaswamy, S. Sapatnekar, P. Banerjee, A framework for exploiting task and data parallelism on distributed memory multicomputers, IEEE Trans. Parallel Distributed Systems 8 (11) (November 1997) 1098–1116.
[42] Remos, http://www-2.cs.cmu.edu/~cmcl/remulac/remos.html, 2003.
[43] U. Rencuzogullari, S. Dwarkadas, Dynamic adaptation to available resources for parallel computing in an autonomous network of workstations, Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'2001), June 2001, pp. 72–81.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 321 of 411

[44] K.C. Sevcik, Characterizations of parallelism in applications and their use in scheduling, Performance Eval. Rev. 17 (1) (May 1989) 171–180.
[45] G. Shao, R. Wolski, F. Berman, Modeling the cost of redistribution in scheduling, Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, March 1997.
[46] H.J. Siegel, J.K. Antonio, R.C. Metzger, M. Tan, Y.A. Li, Heterogeneous computing, in: A.Y. Zomaya (Ed.), Parallel and Distributed Computing Handbook, McGraw-Hill, New York, NY, 1996, pp. 725–761.
[47] H.J. Siegel, T.D. Braun, H.G. Dietz, M.B. Kulaczewski, M. Maheswaran, P.H. Pero, J.M. Siegel, J.E. So, M. Tan, M.D. Theys, L. Wang, The PASM Project: a study of reconfigurable parallel computing, Proceedings of the Second International Symposium on Parallel Architectures, Algorithms, and Networks, June 1996, pp. 529–536.
[48] H.J. Siegel, H.G. Dietz, J.K. Antonio, Software support for heterogeneous computing, in: A.B. Tucker Jr. (Ed.), The Computer Science and Engineering Handbook, CRC Press, Boca Raton, FL, 1997, pp. 1886–1909.
[49] H.J. Siegel, T. Schwederski, W.G. Nation, J.B. Armstrong, L. Wang, J.T. Kuehn, R. Gupta, M.D. Allemang, D.G. Meyer, D.W. Watson, The design and prototyping of the PASM reconfigurable parallel processing system, in: A.Y. Zomaya (Ed.), Parallel Computing: Paradigms and Applications, International Thomson Computer Press, London, UK, 1996, pp. 78–114.
[50] M. Singh, S. Singh, Image segmentation optimisation for X-ray images of airline luggage, Proceedings of the 2004 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS'2004), July 2004, pp. 10–17.
[51] M. Srinivas, L.M. Patnaik, Genetic algorithms: a survey, Computer 27 (6) (June 1994) 17–26.
[52] J. Subhlok, P. Lieu, B. Lowekamp, Automatic node selection for high performance applications on networks, Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'1999), May 1999, pp. 163–172.
[53] M. Tan, H.J. Siegel, A stochastic model for heterogeneous computing and its application in data relocation scheme development, IEEE Trans. Parallel Distributed Systems 9 (11) (November 1998) 1088–1101.
[54] M. Tan, H.J. Siegel, J.K. Antonio, Y.A. Li, Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system, IEEE Trans. Parallel Distributed Systems 8 (8) (August 1997) 857–871.
[55] B. Veeravalli, W.H. Min, Scheduling divisible loads on heterogeneous linear daisy chain networks with arbitrary processor release times, IEEE Trans. Parallel Distributed Systems 15 (3) (March 2004) 273–288.
[56] J.G. Verly, R.L. Delanoy, Model-based automatic target recognition (ATR) system for forwardlooking groundbased and airborne imaging Laser Radars (LADAR), Proc. IEEE 84 (2) (February 1996) 126–163.
[57] Q. Wang, K.H. Cheng, List scheduling of parallel tasks, Inform. Process. Lett. 37 (5) (March 1991) 291–297.
[58] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, J. Parallel Distributed Comput. 47 (1) (November 1997) 8–22.
[59] F.J. Weil, L.H. Jamieson, E.J. Delp, Dynamic intelligent scheduling and control of reconfigurable parallel architectures for computer vision/image processing, J. Parallel Distributed Comput. 13 (3) (November 1991) 273–285.
[60] T. Yang, C. Fu, Heuristic algorithms for scheduling iterative task computations on distributed memory machines, IEEE Trans. Parallel Distributed Systems 8 (6) (June 1997) 608–622.



Yu-Kwong Kwok is an Associate Professor in the Department of Electrical and Electronic Engineering at the University of Hong Kong (HKU). Before joining the HKU in August 1998, he was a Visiting Scholar for 1 year in the Parallel Processing Laboratory at the School of Electrical and Computer Engineering at the Purdue University. He recently served as a Visiting Associate Professor at the Department of Electrical Engineering—Systems at the University of Southern California from August 2004 to July 2005, on his sabbatical leave from HKU. He received his B.Sc. degree in Computer Engineering from the University of Hong Kong in 1991, the M.Phil. and Ph.D. degrees in Computer Science from the Hong Kong University of Science and Technology (HKUST) in 1994 and 1997, respectively. His research interests include distributed computing systems, wireless networking, and mobile computing. He is a Senior Member of the IEEE. He is also a Member of the ACM, the IEEE Computer Society, and the IEEE Communications Society. He received the Outstanding Young Researcher Award from HKU in November 2004.



Anthony A. Maciejewski received his B.SEE., M.S., and Ph.D. degrees from the Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at the Purdue University, West Lafayette. He is currently the Head of the Department of Electrical and Computer Engineering at the Colorado State University. He is a Fellow of the IEEE. A complete vita is available at: www.engr.colostate.edu/∼aam.



H. J. Siegel was appointed the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at the Colorado State University (CSU) in August 2001, where he is also a Professor of Computer Science. In December 2002, he became the first Director of the University-wide CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a Professor at the Purdue University. He received two B.S. degrees from MIT, and the M.A., M.S.E., and Ph.D. degrees from the Princeton University. Prof. Siegel has coauthored over 300 published papers on parallel and distributed computing and communication. He is a Fellow of the IEEE Fellow and a Fellow of the ACM. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of six international conferences, and Chair/Co-Chair of five workshops. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/∼hj.



Ishfaq Ahmad received his B.Sc. degree in Electrical Engineering from the University of Engineering and Technology, Lahore, Pakistan, in 1985, and a M.S. degree in Computer Engineering and a Ph.D. degree in Computer Science from the Syracuse University, New York, USA, in 1987 and 1992, respectively. His recent research focus has been on developing parallel programming tools, scheduling and mapping algorithms for scalable architectures,

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 322 of 411

heterogeneous computing systems, distributed multimedia systems, video compression techniques, and grid computing. His research work in these areas is published in over 150 technical papers in refereed journals and conferences. He is currently a Full Professor of Computer Science and Engineering in the CSE Department of the University of Texas at Arlington. AT UTA, he leads IRIS (Institute for Research In Security), a multi-disciplinary research center engaged in safety and security related technologies. He is an Associate Editor of Cluster Computing, Journal of Parallel and Distributed Computing, IEEE Transactions on Circuits and Systems for Video Technology, IEEE Concurrency, and IEEE Distributed Systems Online.



Arif Ghafoor is currently a Professor in the School of Electrical and Computer Engineering at the Purdue University, West Lafayette, IN, and is the Director of Information Infrastructure Security Research Laboratory and Distributed Multimedia Systems Laboratory. He has been actively engaged in research areas related to database security, parallel and distributed computing, and multimedia information systems. He has published numerous technical papers in these areas. He has served on the editorial boards of various journals. He is a Fellow of the IEEE. He has received the IEEE Computer Society 2000 Technical Achievement Award for his research contributions in the area of multimedia systems.

View publication stats

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 323 of 411

CHAPTER 6

Mapping Tasks onto Distributed Heterogeneous Computing Systems Using a Genetic Algorithm Approach

MITCHELL D. THEYS
University of Illinois at Chicago
TRACY D. BRAUN, HOWARD JAY SIEGEL, and ANTHONY A. MACIEJEWSKI,
Purdue University
YU-KWONG KWOK
The University of Hong Kong

6.1 INTRODUCTION

Different portions of an application task often require different types of computation. In general, it is impossible for a single machine architecture with its associated compiler, operating system, and programming tools to satisfy all the computational requirements in such an application equally well. One type of *heterogeneous computing* (*HC*) environment consists of a suite of different types of machines, high-speed interconnections, interfaces, operating systems, communication protocols, and programming environments that provide a variety of architectural capabilities. Such an HC environment can be orchestrated to perform an application that has diverse execution requirements [17, 20, 22, 32, 46, 47, 50]. An application task can be decomposed into *subtasks*, where each subtask is computationally homogeneous (well-suited to a single machine), and different subtasks may have different machine architectural requirements. These subtasks may have data dependencies among them. A group of independent tasks, known as a *meta-task*, can also be executed in the HC environment. The tasks in a *meta-task* have no data dependencies among them, and may have different architectural requirements.

Solutions to Parallel and Distributed Computing Problems, Edited by A. Zomaya, F. Ercal, and S. Olariu
ISBN 0-471-35352-3 © 2001 John Wiley & Sons, Inc.

Darell Long, Ph.D.
09/17/2025
Exhibit H
H. Bautista, CSR #11600

MICROSOFT CORP.
EXHIBIT 1008

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 324 of 411

Once the set of subtasks or independent tasks to be executed is known, the following decisions must be made: *matching*, i.e., assigning subtasks or independent tasks to machines, and *scheduling*, i.e., ordering subtask or independent task execution for each machine and (for subtasks) ordering intermachine data transfers. The resulting matching and scheduling scheme is called a *mapping*. In the studies reported here, the primary goal of HC is to find a mapping that will achieve the minimal *completion time*, i.e., the minimal overall execution time of the application task or meta-task on the machine suite.

It is well known that such a mapping problem is, in general, NP complete [19, 29]. A number of approaches to different aspects of this problem have been proposed (e.g., [18, 20, 30, 38, 51, 55]). These approaches include both static and dynamic heuristics. *Static* heuristics are executed off-line, in a planning mode, when the application task or meta-task to map is known in advance. *Dynamic* heuristics are executed on-line, in real-time, and can make use of feedback from the HC system (e.g., [37]). Heuristics developed to perform the mapping function are often difficult to compare because of different underlying assumptions in the original studies of each heuristic (e.g., number and types of machines in the HC suite) [5]. Various researchers have considered the use of genetic-algorithm-based approaches for this mapping function (e.g., [45, 48, 52, 61]). This chapter summarizes three particular genetic-algorithm-based approaches, each for a different HC environment.

Two static approaches and one dynamic approach used for solving the mapping problems for different HC environments are discussed. The first approach decides the subtask to machine assignments, orders the execution of the subtasks assigned to each machine, and schedules the data transfers among subtasks [58]. The characteristics of this approach include separation of the matching and the scheduling representations, independence of the chromosome structure from the details of the communication subsystem, and consideration of overlap among all computations and communications that obey subtask precedence constraints. The computation and communication overlap is limited only by intersubtask data dependencies and machine/network availability. This genetic-algorithm-based approach can be applied to performing the mapping in a variety of HC systems. It is applicable to the static mapping of production jobs and can be readily used to collectively schedule a set of tasks that are decomposed into subtasks.

The second approach focuses on a particular application domain [*iterative automatic target recognition* (*ATR*) tasks] and an associated specific class of dedicated heterogeneous parallel hardware platforms. For the computational environment considered, a methodology for real-lime, on-line, input-data-dependent remapping of the application subtasks to the processors (machines) in the heterogeneous parallel hardware platform using a previously stored off-line, statically determined, mapping is presented [8, 34, 35]. The operating system makes a heuristic-based decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the operating system selects a previously determined and stored genetic-algorithm-based mapping that is appropriate for the given state of the application (e.g., the number of objects it is currently tracking).

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 325 of 411

The third approach examines the mapping of meta-tasks to machines in an HC suite. It is assumed in this approach that each machine executes a single task at a time, in the order in which the tasks arrive, and there are no dependencies among the tasks. Because of these assumptions, scheduling is simplified and the resulting solutions of the mapping heuristics focus more on finding an efficient matching of tasks to machines. Eleven different static mapping heuristics from the literature were implemented and compared by simulation studies under one set of common assumptions [6]. One of these heuristics was a genetic algorithm, providing insight into the relative performance of genetic algorithms to other mapping techniques.

The organization of this chapter is as follows. The details of the problem and assumptions made for each of the approaches are presented in Section 6.2. Section 6.3 briefly describes the basic steps of any genetic algorithm. In Section 6.4, a summary of the results from the static subtask mapping approach is presented [58]. Section 6.5 discusses results obtained from the on-line use of off-line-derived mappings approach [8, 34]. Finally, Section 6.6 details experiences from implementing the static meta-task mapping approach [6].

The research reported in this chapter was supported in part by the DARPA/ITO Quorum Program project called *MSHN* (Management System for Heterogeneous Networks) [26]. MSHN is a collaborative research effort among the Naval Postgraduate School, NOEMIX, Purdue University, and the University of Southern California. The technical objective of the MSHN project is to design, prototype, and refine a distributed resource management system that leverages the heterogeneity of resources and tasks to deliver the requested quality of service. The heuristics developed in this chapter or their derivatives may be included in the MSHN prototype.

6.2 PROBLEM DESCRIPTIONS

6.2.1 Static Matching and Scheduling of Subtasks

There are many open research problems in the HC field [36, 46]. To isolate and focus on the mapping problem, researchers typically make assumptions about other components of an overall HC system (e.g., [45, 48]). This subsection considers the assumptions made in [58] for the static mapping of subtasks.

It is assumed that the application task is written in some machine-independent language (e.g., [59]). It is also assumed that an application task is decomposed into multiple subtasks and the data dependencies among them are known and are represented by a *directed acyclic graph* (*DAG*). If intermachine data transfers are data dependent, then some set of expected data transfers must be assumed. The estimated expected execution time for each subtask on each machine is assumed to be known a priori. Finding the estimated expected execution times for subtasks is another research problem, which is beyond the scope of this chapter. Approaches based on task profiling and analytical benchmarking are surveyed in [32, 46, 47]. The HC system is assumed to have operating system support for executing each

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 326 of 411

subtask on the machine it is assigned and for performing intermachine data transfers as scheduled by this genetic-algorithm-based approach.

In the type of HC system considered here, an application task is decomposed into a set of subtasks S. Define $|S|$ to be the number of subtasks in the set, and s_i to be the ith subtask $0 \leq i < |S|$. An HC environment consists of a set of machines M. Define $|M|$ to be the number of machines and m_j to be the jth machine, $0 \leq j < |M|$. The *global data items* (*gdis*), i.e., data items that need to be transferred between subtasks, form a set G. Define $|G|$ to be the number of items and gdi_k to be the kth global data item, $0 \leq k < |G|$.

It is assumed that for each global data item there is a single subtask that produces it (*producer*), and there are some subtasks that need this data item (*consumers*). Hence, the task is represented by a *single-producer directed acyclic graph* (*SPDAG*). Each edge goes from a producer to a consumer and is labeled by the gdi that is transferred over it. Figure 6.1 shows an example SPDAG.

The following further assumptions are made for the static subtask mapping problem. One is the exclusive use of the HC environment for the application; and that the genetic-algorithm-based mapper is in control of the HC machine suite. Another is nonpreemptive subtask execution. Also, all input data items of a subtask must be received before its execution can begin, and none of its output data items are available until the execution of this subtask is finished. If a data conditional is based on input data, it is assumed to be contained inside a subtask. A loop that uses an input data item to determine one or both of its bounds is also assumed to be contained inside a subtask. These restrictions help make the mapping problem more manageable, and solving this problem under these assumptions is a significant step toward solving the general mapping problem.

For any static heuristic, it is assumed that an accurate estimate of the expected execution time for each subtask on each machine is known prior to execution and contained within an $|M| \times |S|$ ETC (expected time to compute) *matrix*. $\text{ETC}(i, j)$ is the estimated execution time for subtask s_i on machine m_j.

These times are assumed to include the time to move the executables and initial data (not gdis) associated with subtask s_i to machine m_j when necessary.] The

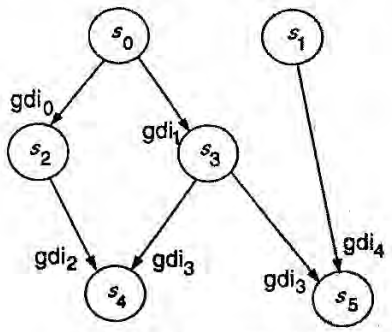


Figure 6.1 An example SPDAG.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 327 of 411

assumption that these estimated expected execution times are known is commonly made when studying mapping heuristics for HC systems (e.g., [23, 31, 48]).

6.2.2 Semistatic Matching and Scheduling of Subtasks

In [7, 8], a detailed design of an *intelligent operating system (IOS)* is proposed for a particular application domain in which (1) an iterative application is to be mapped onto an associated specific type of dedicated heterogeneous parallel hardware platform; and (2) the execution of each iteration can be represented by a DAG of subtasks. To minimize the execution time of such an iterative application in a heterogeneous parallel computing environment, an appropriate mapping scheme is needed. However, when some of the characteristics of the application subtasks are unknown a priori and will change from iteration to iteration during execution-time, it may not be feasible or desirable to use the same off-line-derived mapping throughout the whole execution of the application.

In such situations, a semistatic methodology [7, 8, 35] can be employed that starts with an initial mapping, but dynamically decides whether to remap the application with a mapping previously determined off-line. This is done by observing, from one iteration to another, the effects of the changing characteristics of the application's input data, called *dynamic parameters*, on the application's execution time. That is, the IOS will be able to make a heuristically determined decision during the execution of the application whether to perform a remapping based on information generated by the application from its input data. If the decision is to remap, the IOS will be able to select a precomputed and stored mapping that is appropriate for the given state of the application (e.g., the number of objects the ATR system is currently tracking). This remapping process will, in general, require a certain system *reconfiguration time* for relocating the data and program modules.

The semistatic method differs considerably from other real-time HC mapping techniques in that it involves the on-line, real-time use of off-line, precomputed mappings. This is significant because it is possible for off-line heuristics to have much longer execution times to search for a good solution than what is practical for an on-line heuristic. Thus, with the semistatic method, the mapping quality of an off-line, time-consuming heuristic can be approached at real-time speeds. As detailed in Section 6.5.6, the GA described in Section 6.4 is enhanced to be used as an off-line heuristic for determining high-quality mappings for on-line use. In an extensive simulation study, it was found that the semistatic method is much more effective than a purely dynamic approach.

6.2.3 Static Matching and Scheduling for Meta-Tasks

This subsection considers the assumptions made in [6] for the static mapping of meta-tasks. Recall from Section 6.1 that a meta-task is defined as a collection of independent tasks with no data dependencies (a given task, however, may have subtasks and dependencies among the subtasks). For this case study, it is assumed that static (i.e., off-line or predictive) mapping of meta-tasks is being performed. (In

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 328 of 411

some systems, all tasks and subtasks in a meta-task, as defined earlier, are referred to as just tasks.)

It is also assumed that each machine executes a single task at a time in the order in which the tasks arrived. Because there are no dependencies among the tasks, scheduling is simplified, and thus the resulting solutions of the mapping heuristics focus more on finding an efficient matching of tasks to machines. It is also assumed that the size of the meta-task T (number of tasks to execute) is $|T|$, and the number of machines in the HC environment is $|M|$, and both are static and known a priori. Expected task execution times are specified in an $|M| \times |T|$ ETC matrix, analogous to the one defined in Section 6.2.1 for subtasks. It is assumed that the HC system is dedicated for the meta-task, and controlled by the mapper.

6.3 GENETIC ALGORITHM OVERVIEW

Genetic algorithms (*GAs*) are a useful heuristic approach to finding near-optimal solutions in large search spaces [14, 25, 27]. There is a large variety of approaches to GAs; many are surveyed in [40, 49]. The following is a brief overview of GAs to provide background for the description of the proposed approaches.

The first step necessary to employ a GA is to encode some of the possible solutions to the optimization problem as a set of strings (*chromosomes*). Each chromosome represents one solution to the problem, and a set of chromosomes is referred to as a *population*. The next step is to derive an initial population. A random set of chromosomes is often used as the initial population. Some specified chromosomes can also be included as *seeds*. This initial population is the first generation from which the evolution begins.

The third step is to evaluate the quality of each chromosome. Each chromosome is associated with a *fitness value*, which in this case is the completion time of the solution (mapping) represented by this chromosome (i.e., the expected execution time of the application task or meta-task if the mapping specified by this chromosome were used). The objective of the GA search is to find a chromosome that has the optimal fitness value. The *selection* process is the next step. In this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. The population size is typically kept constant.

Selection is followed by the *crossover* step. With some probability, pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in population can be *mutated* with some probability. The mutation process transforms a chromosome into another valid chromosome that may or may not already be in the current population. The new population is then evaluated. If none of the stopping criteria has been met, the new population goes through another cycle (iteration) of selection, crossover, mutation, and evaluation. These cycles continue until one of the stopping criteria is met.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 329 of 411

```
GA_matching_scheduling {
    initial population generation;
    evaluation;
    while (stopping criteria not met) {
        selection;
        crossover;
        mutation;
        evaluation;
    }
    output the best solution found;
}
```

Figure 6.2 General procedure for a genetic algorithm, based on [49].

In summary, the following items must be determined to implement a GA for a given optimization problem: (1) an encoding, (2) an initial population, (3) an evaluation using a particular fitness function, (4) a selection mechanism, (5) a crossover mechanism, (6) a mutation mechanism, and (7) a set of stopping criteria. The outline of the GA-based approach is shown in Fig. 6.2. Details of three GA-based approaches will be discussed in the following sections.

6.4 STATIC MATCHING AND SCHEDULING OF SUBTASKS

6.4.1 Introduction

This section discusses the GA-based approach found in [58] for the static mapping of subtasks. This section presents the details about the chromosome representation used, how population generation was performed, the mutation and crossover operators used, and comparisons with nonevolutionary approaches.

6.4.2 Chromosome Representation

Each chromosome consists of two parts: the matching string and the scheduling string. Let *mat* be the *matching string*, which is a vector of length $|S|$, such that $\text{mat}(i) = m_j$, where $0 \leq i < |S|$ and $0 \leq j < |M|$, i.e., subtask s_i is assigned to machine m_j.

The *scheduling string* (*ss*) is a topological sort [12] of the SPDAG, i.e., a total ordering of the nodes (subtasks) in the SPDAG that obeys the precedence constraints. Define ss to be the scheduling string, which is a vector of length $|S|$, such that $\text{ss}(k) = s_i$, where $0 \leq i, k < |S|$, and each s_i appears only once in the vector, i.e., subtask s_i is the kth subtask in the scheduling string. Because it is a topological sort, if ss(k) is a consumer of a global data item produced by ss(j), then $j < k$. The scheduling string gives an ordering of the subtasks that is used by the evaluation step.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 330 of 411

Then in this GA-based approach, a chromosome is represented by a two-tuple [mat,ss]. Thus, a chromosome represents the subtask-to-machine assignments (matching) and the execution ordering of the subtasks assigned to the same machine. The scheduling of the global data item transfers and the relative ordering of subtasks assigned to different machines are determined by the evaluation step. Figure 6.3 illustrates two different chromosomes for the SPDAG in Fig. 6.1, for $|S| = 6$, $|M| = 3$, and $|G| = 5$.

6.4.3 Initial Population Generation

In the initial population generation step, a predefined number of chromosomes are generated, the collection of which forms the initial population. When generating a chromosome, a new matching string is obtained by randomly assigning each subtask to a machine. To form a scheduling string, the SPDAG is first topologically sorted to form a basis scheduling string. Then, for each chromosome in the initial population, this basis string is mutated a random number of times (between one and the number of subtasks) using the scheduling-string mutation operator to generate the ss vector (which is a valid topological sort of the given SPDAG). Furthermore, it is common in GA applications to incorporate solutions from some nonevolutionary heuristics into the initial population, which may reduce the time needed for finding a satisfactory solution [14]. In this GA-based approach, along with those chromosomes

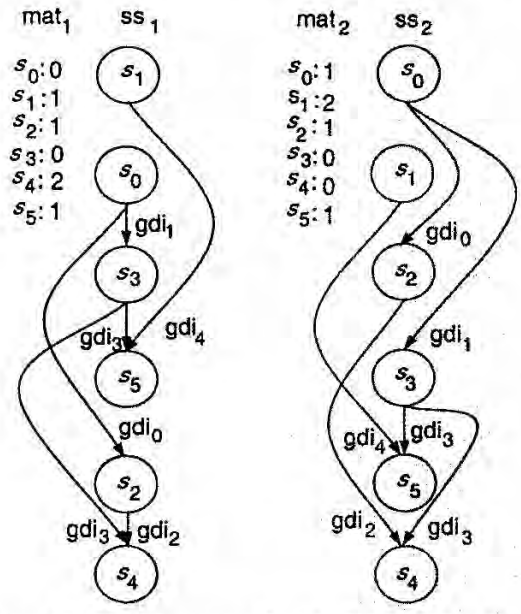


Figure 6.3 Two chromosomes from the SPDAG in Fig. 6.1.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 331 of 411

representing randomly generated solutions, the initial population also includes a seed chromosome that represents the solution from a nonevolutionary baseline heuristic (see Section 6.4.8).

Each newly generated chromosome is checked against those previously generated. If a new chromosome is identical to any of the existing ones, it is discarded and the process of chromosome generation is repeated until a unique new chromosome is obtained.

6.4.4 Selection

In this step, the chromosomes in the population are first ordered (*ranked*) by their fitness values from the best to the worst. Those having the same fitness value are ranked arbitrarily among themselves. Then a *rank-based roulette wheel selection scheme* is used to implement the selection step [27, 49]. In the rank-based selection scheme, each chromosome is allocated a sector on a roulette wheel. Let P denote the population size and A_i denote the angle of the sector allocated to the ith-ranked chromosome. The zeroth-ranked chromosome is the fittest and has the sector with the largest angle A_0, whereas the $(P-1)$-th-ranked chromosome is the least fit and has the sector with the smallest angle A_{P-1}. The ratio of the sector angles between two adjacently ranked chromosomes is a constant $R = A_i/A_{i+1}$, where $0 \leq i < P-1$. If the 360 degrees of the wheel are normalized to one, then $A_i = R^{P-i-1} \times (1-R)/(1-R^P)$, where $R > 1$, $0 \leq i < P$, and $0 < A_i < 1$.

The selection step generates P random numbers, ranging from zero to one. Each number falls in a sector on the roulette wheel and a copy of the owner chromosome of this sector is included in the next generation. Because a better solution has a larger sector angle than that of a worse solution, there is a higher probability that one or more copies of this better solution will be included in the next generation. In this way, the population for the next generation is determined. Thus, the population size is always P, and it is possible to have multiple copies of the same chromosome.

This GA-based approach also incorporates *elitism* [41]. At the end of each iteration, the best chromosome is always compared with the previous best (elite) chromosome, a copy of which is stored separately from the population. If the best chromosome is better than the elite chromosome, a copy of it becomes the elite chromosome. If the best chromosome is not as good as the elite chromosome, a copy of the elite chromosome replaces the worst chromosome in the population. Elitism is important because it guarantees that the quality of the best solutions found over generations is monotonically nondecreasing.

6.4.5 Crossover Operators

The crossover operator selects a random number of pairs of scheduling strings, where every string has an equal probability of being selected. For each pair, it randomly generates a cutoff point, which divides the scheduling strings of the pair into top and bottom parts. Then, the subtasks in each bottom part are reordered. The new ordering of the subtasks in one bottom part is the relative positions of these

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 332 of 411

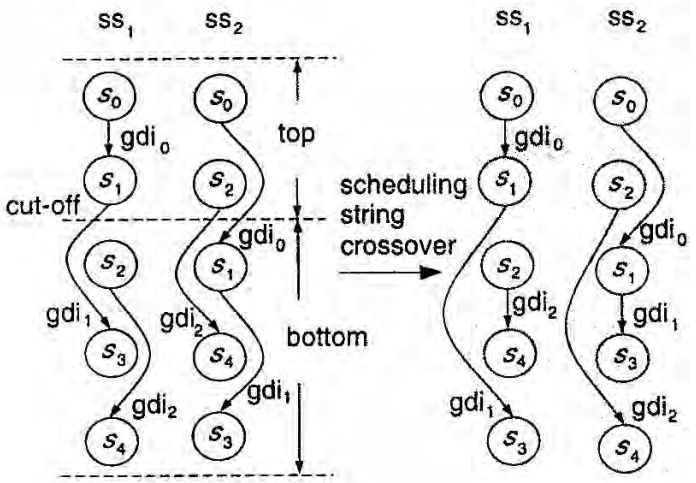


Figure 6.4 A scheduling string crossover example.

subtasks in the other original scheduling string in the pair, thus guaranteeing that the newly generated scheduling strings (which replace the originals) are valid schedules. Figure 6.4 demonstrates such a scheduling string crossover process.

The crossover operator selects a random number of pairs of matching strings, where every string has an equal probability of being selected. For each pair, it randomly generates a cutoff point to divide both matching strings of the pair into two parts. Then the machine assignments of the bottom parts are exchanged, and the new strings replace the originals.

6.4.6 Mutation Operators

The scheduling-string mutation operator selects a random number of scheduling strings, where every string has an equal probability of being selected. Then for each chosen scheduling string, it randomly selects a victim subtask. The *valid range* of the victim subtask is the set of the positions in the scheduling string at which this victim subtask can be placed without violating any data dependency constraints. Specifically, the valid range is after all source subtasks of the victim subtask and before any destination subtask of the victim subtask. After a victim subtask is chosen, it is moved randomly to another position in the scheduling string within its valid range. The new string replaces the original. Figure 6.5 shows an example of this mutation process. For the matching string mutation operator, every matching string has an equal probability of having one of its subtasks assigned to a different, randomly selected, machine.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 333 of 411

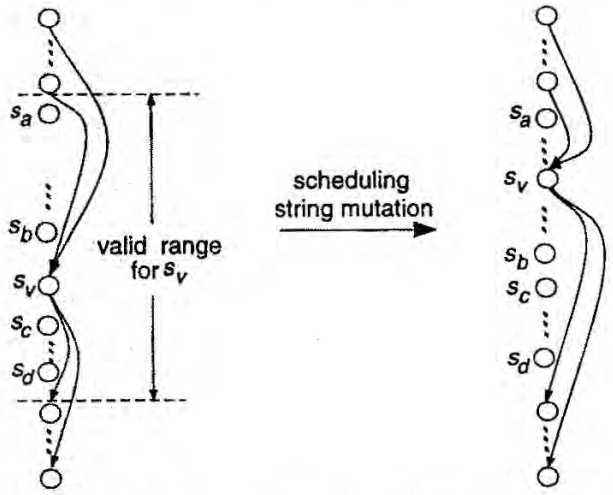


Figure 6.5 A scheduling string mutation example. Only edges to and from the victim subtask s_v are shown. Before the mutation, s_v is between s_b and s_c. After the mutation, it is moved to between s_a and s_b.

6.4.7 Evaluation

The final step of an iteration of a GA is the evaluation of each chromosome to determine its fitness value. In this GA-based approach, the chromosome structure is independent of any particular communication subsystem. Only the evaluation step needs the communication characteristics of the given HC system to schedule the data transfers.

To test the effectiveness of this GA-based approach, an example communication system modeled after a HiPPI LAN with a central crossbar switch [28, 53] is assumed to connect a suite of machines. Each machine in the HC suite has one input data link and one output data link. All these links are connected to a central crossbar switch. If a subtask needs a gdi that is produced or consumed earlier by a different subtask on the same machine, the communication time for this item is zero. Otherwise, the communication time is obtained by dividing the size of the gdi by the smaller bandwidth of the output link of the source machine and the input link of the destination machine. In this research, it is assumed that for a given machine the bandwidths of the input link and the output link are equal to each other. It is also assumed that the crossbar switch has a higher bandwidth than that of each link. The communication latency between any pair of machines is assumed to be the same. Data transfers are neither preemptive nor multiplexed. Once a data-transfer path is established, it cannot he relinquished until the data item (i.e., some gdi_k) scheduled to be transferred over this path is received by the destination machine. Multiple data transfers over the same path must be serialized.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 334 of 411

In the evaluation step, for each chromosome the final order of execution of the subtasks and the intermachine data transfers are determined. The evaluation procedure considers the subtasks in the order they appear on the scheduling string. Subtasks assigned to the same machine are executed exactly in the order specified by the scheduling string. For subtasks assigned to different machines, the actual execution order may deviate from that specified by the scheduling string due to factors such as input-data availability and machine availability. This is explained below.

Before a subtask can he scheduled, all of its input gdis must be received. For each subtask, its input data items are considered by the evaluation procedure in the order of their producers' relative positions in the scheduling string. In Fig. 6.6, a simple example is shown to illustrate the evaluation for a given chromosome. In this example (as well as some others given later), because there are only two machines, the source and destination machines for the gdi transfers are implicit.

When a subtask to be scheduled has multiple input gdis that have not been received, the gdi whose producer subtask is listed earliest in the scheduling string is considered first. The reason for this ordering is to attempt to better utilize the overlap of subtask executions and intermachine data communications. The following example illustrates this idea. Let $ss(0) = s_0$, $ss(1) = s_1$, $ss(2) = s_2$, as shown in Fig. 6.7*a*. Let s_2 need two gdis, gdi_0 and gdi_1, from s_0 and s_1, respectively. Depending on the subtask to machine assignments, the data transfers of gdi_0, and gdi_1 could be either local, i.e., within a machine, or across machines. If at least one data transfer is local, then the scheduling is trivial because it is assumed that local transfers within a machine take negligible time. However, there exist two situations where both data transfers are across machines so that they need to be ordered.

Situation 1 Let s_0 and s_1 be assigned to the same machine m_0 and s_2 be assigned to another machine m_1, as shown in Fig. 6.7*b*. In this situation, because s_0 is to be executed before s_1, gdi_0 is available before gdi_1 becomes available on machine m_0. Thus, it is better to consider the gdi_0 transfer before the gdi_1 transfer.

Situation 2 Let the three subtasks s_0, s_1, and s_2 be assigned to three different machines m_0, m_1, and m_2, as shown in Fig. 6.7*c*. In this situation, if there is a data dependency from s_0 to s_1, then s_0 finishes its execution before s_1 could start. Therefore, gdi_0 is available before gdi_1 becomes available. Hence, it is better to consider the gdi_0 transfer before the gdi_1 transfer. If there are no data dependencies from s_0 to s_1, the gdi_0 transfer can still be considered before the gdi_1 transfer. While this may not be necessary in this case, it is reasonable to do because there may be some other chromosome(s) that have $ss(0) = s_1$ and $ss(1) = s_0$. When such a chromosome is evaluated, the gdi_1 transfer will be considered before the gdi_0 transfer. Therefore, it is possible for all input gdi scheduling orderings for gdi_0 and gdi_1 to be examined.

Data forwarding is another important feature of this evaluation process. For each input data item to be considered, the evaluation process chooses the source subtask

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 335 of 411

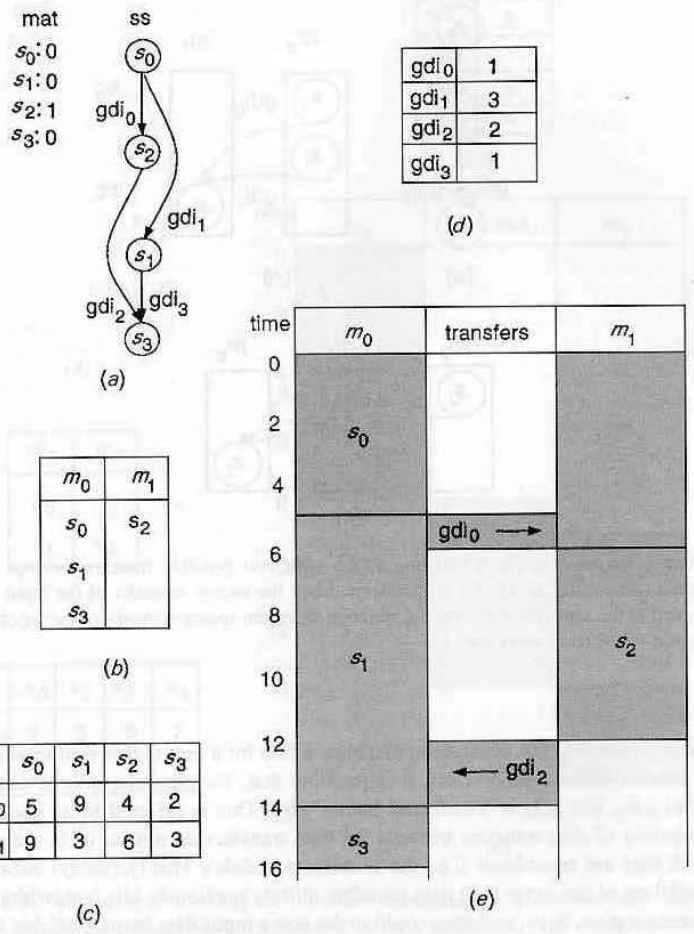


Figure 6.6 A very simple example showing the evaluation step: (*a*) the chromosome; (*b*) the subtask execution ordering on each machine given by (*a*); (*c*) the estimated subtask execution times; (*d*) the gdi intermachine transfer times (transfers between subtasks assigned to the same machine take zero time); and (*e*) the subtask execution and data-transfer timings, where the completion time for this chromosome is 16.

from among the producer of this data item and all the consumers that have received this data item. These consumers are *forwarders* [51]. The one (either the producer or a forwarder) from which the destination subtask will receive the data item at the earliest possible time is chosen.

After the source subtask is chosen, the data transfer for the input data item is scheduled. A transfer starts at the earliest point in time when the path from the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 336 of 411

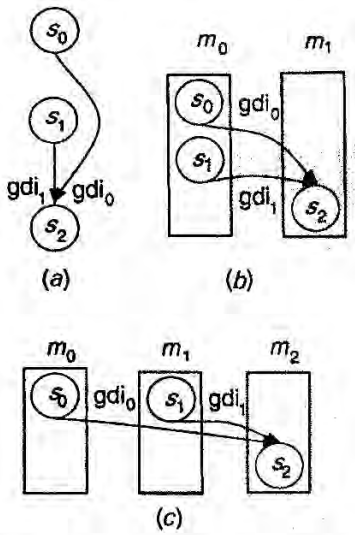


Figure 6.7 An example scheduling string with two possible matching strings: (*a*) the example scheduling string; (*b*) the situation when the source subtasks of the input gdis are assigned to the same machine; (*c*) the situation when the source subtasks of the input gdis are assigned to different machines.

source machine to the destination machine is free for a period that is at least equal to the needed transfer time. Thus, it is possible that, for example, gdi_1 is considered before gdi_2, but gdi_2 is transferred before gdi_1. This is referred to as *out-of-order* scheduling of data transfers because the data transfers do not occur in the order in which they are considered (i.e., the *in-order* schedule). This (possibly) out-of-order scheduling of the input item data transfers utilizes previously idle bandwidths of the communication links, and thus could make some input data items available to some subtasks earlier than the in-order scheduling. As a result, some subtasks could start their execution earlier, which would in turn decrease the overall task completion time. Figures 6.8 and 6.9 show the in-order scheduling and the out-of-order scheduling for the same chromosome, respectively. In the in-order scheduling, the transfer of gdi_1 is scheduled before the transfer of gdi_2 because subtask s_2 input data transfers are considered before those of subtask s_3. In this example, the out-of-order schedule does decrease the total execution time of the given task.

When two chromosomes have different matching strings, they are different solutions because the subtask-to-machine assignments are different. However, two chromosomes that have the same matching string but different scheduling strings may or may not represent the same solution. This is because the scheduling-string information is used in two cases: (1) for scheduling subtasks that have been assigned

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 337 of 411

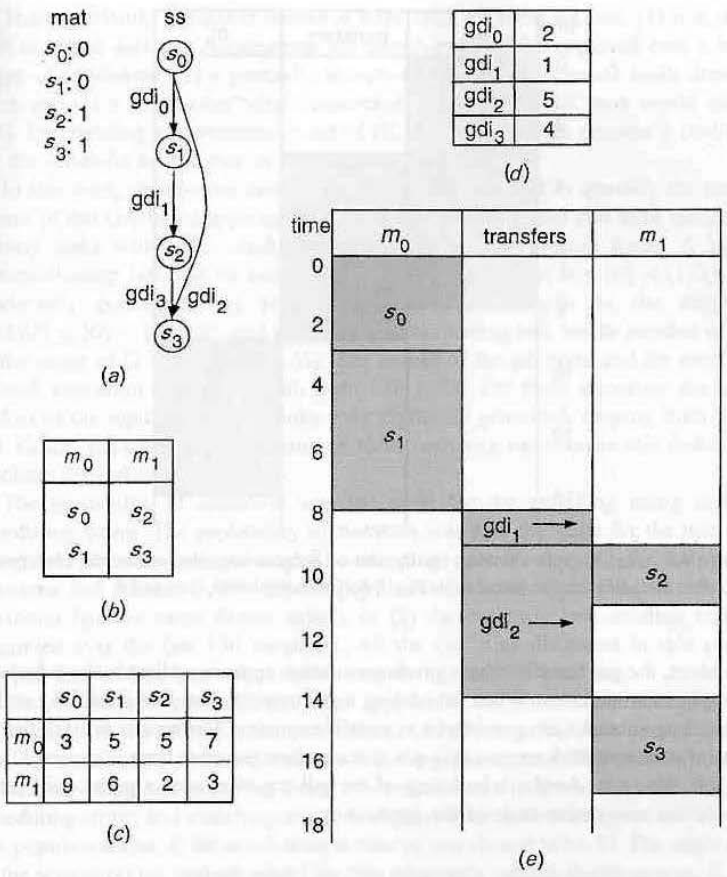


Figure 6.8 An example showing the in-order scheduling of a chromosome: (*a*) the chromosome; (*b*) the subtask execution ordering on each machine; (*c*) the estimated subtask execution times; (*d*) the gdi transfer times (transfers between subtasks assigned to the same machine take zero time); and (*e*) the subtask execution and data-transfer timings using in-order transfers (the gdi_1 transfer occurs before the gdi_2 transfer), where the completion time is 17.

to the same machine, and (2) for examining data transfers. Two different scheduling strings could result in the same ordering for (1) and (2).

After a chromosome is evaluated, it is associated with a fitness value, which is the time when the last subtask finishes its execution. That is, the fitness value of a chromosome is the overall execution time of the task, given the mapping decision specified by this chromosome and by the evaluation process.

In summary, this evaluation mechanism considers subtasks in the order in which they appear in the scheduling string. For a subtask that requires some gdis from other

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 338 of 411

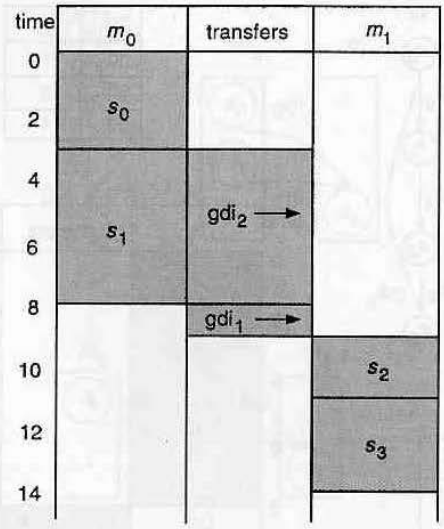


Figure 6.9 An example showing the the out-of order scheduling, where the chromosome and other statistics are the same as in Fig. 6.8. The completion time is 14.

machines, the gdi transfer whose producer subtask appears earliest in the scheduling string is examined first. When scheduling a gdi transfer, both the producing and the forwarding subtasks are considered as possible sources. The source subtask that lets this consumer subtask receive this gdi at the earliest possible time is chosen to send the gdi. The out-of-order scheduling of the gdi transfers over a path could further reduce the completion time of the application.

6.4.8 Experimental Results

To measure the performance of this GA-based approach, randomly generated scenarios were used, where each *scenario* corresponded to an SPDAG, the associated subtask execution times, the sizes of the associated gdis, and the communication-link bandwidths of the machines. The scenarios were generated for different numbers of subtasks and different numbers of machines, as specified below. The estimated expected execution time for each subtask on each machine, the number of gdis, the size of each gdi, and the bandwidth of each input link of each machine were randomly generated with uniform probability over some predefined ranges. For each machine, the bandwidth of the output link is made equal to that of the input link. The producer and consumers of each gdi were also generated randomly. The scenario generation used a $|G| \times |S|$ dependency matrix to guarantee that the precedence constraints from data dependencies were acyclic.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 339 of 411

These randomly generated scenarios were used for three reasons: (1) it is desirable to obtain data that demonstrate the effectiveness of the approach over a broad range of conditions, (2) a generally accepted set of HC benchmark tasks does not exist, and (3) it is not clear what characteristics a "typical" HC task would exhibit [56]. Determining a representative set of HC task benchmarks remains a challenge for the scientific community in this research area.

In this work, small-scale and larger scenarios were used to quantify the performance of this GA-based approach. The scenarios were grouped into three categories, namely tasks with light, moderate, and heavy communication loads. A lightly communicating task has its number of gdis in the range of $0 \leq |G| < (1/3)|S|$; a moderately communicating task has its number of gdis in the range of $(1/3)|S| \leq |G| < (2/3)|S|$; and a heavily communicating task has its number of gdis in the range of $(2/3)|S| \leq |G| < |S|$. The ranges of the gdi sizes and the estimated subtask execution times were both from 1 to 1,000. For these scenarios, the bandwidths of the input and output links were randomly generated, ranging from 0.5 to 1.5. Hence, the communication times in these scenarios were source and destination machine dependent.

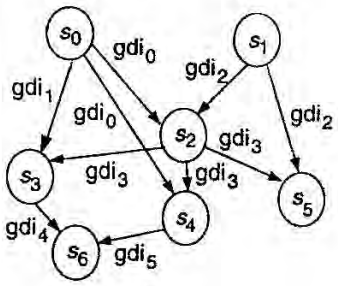
The probability of crossover was the same for the matching string and the scheduling string. The probability of mutation was also the same for the matching string and the scheduling string. The stopping criteria were (1) the number of iterations had reached 1,000, (2) the population had converged (i.e., all the chromosomes had the same fitness value), or (3) the currently best solution had not improved over the last 150 iterations. All the GA runs discussed in this section stopped due to their best solutions not improving for 150 iterations.

The GA-based approach was first applied to 20 small-scale scenarios that involved up to ten subtasks, three machines, and seven global data items. The GA runs for small-scale scenarios had the following parameters. The probabilities for scheduling-string and matching-string crossovers, were both chosen to be 0.4, and scheduling-string and matching-string mutations were both chosen to be 0.1. The GA population size, P, for small-scale scenarios was chosen to be 50. The angle ratio of the sectors on the roulette wheel for two adjacently ranked chromosomes, R, was chosen to be $1 + 1/P$. By using this simple formula, the angle ratio between the slots of the best and median chromosomes for $P = 50$ (and also for $P = 200$ for larger scenarios discussed later in this section) was very close to the optimal empirical ratio value of 1.5 in [60].

The results from a small-scale scenario are used here to illustrate the search process. This scenario had $|S| = 7$, $|M| = 3$, and $|G| = 6$. The SPDAG, the estimated execution times, and the transfer times of the global data items are shown in Fig. 6.10. The total numbers of possible different matching strings and different valid scheduling strings (i.e., topological sorts of the SPDAG) were $3^7 = 2187$ and 16, respectively. Thus, the total search space had $2187 \times 16 = 34{,}992$ possible chromosomes.

Exhaustive searches were performed to find the optimal solutions for the small-scale scenarios. For each of the small-scale scenarios that were conducted, the GA-based approach found one or more optimal solutions that had the same completion

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 340 of 411



(*a*)

	s_0	s_1	s_2	s_3	s_4	s_5	s_6
m_0	872	251	542	40	742	970	457
m_1	898	624	786	737	247	749	451
m_2	708	778	23	258	535	776	15

(*b*)

	m_0m_1	m_0m_2	m_1m_2
gdi_0	489	321	489
gdi_1	1244	818	1244
gdi_2	62	41	62
gdi_3	830	545	830
gdi_4	387	255	387
gdi_5	999	656	999

(*c*)

Figure 6.10 A small-scale simulation scenario: (*a*) the SPDAG; (*b*) the estimated execution times; and (*c*) the transfer times of the gdis from a given source machine to a given destination machine.

time, verified by the best solution(s) found by the exhaustive search. The GA search for a small-scale scenario that had ten subtasks, three machines, and seven gdis took about one minute to find multiple optimal solutions on a Sun Sparc5 workstation, while the exhaustive search took about eight hours to find these optimal solutions.

The performance of this GA-based approach was also examined using larger scenarios with up to 100 subtasks and 20 machines. These larger scenarios were generated using the same procedure as for generating the small scenarios except that the GA population size for larger scenarios was chosen to be 200.

Larger scenarios are intractable problems. It is currently impractical to directly compare the quality of the solutions found by the GA-based approach for these larger scenarios with those found by exhaustive searches. It is also often difficult to compare the performance of different HC task mapping approaches due to the different HC system models that are assumed by researchers when they design mapping heuristics. However, the model used in [30] is similar to the one being used in this research work. Hence, the performance of the GA-based approach on larger scenarios was compared with the nonevolutionary *levelized min-time* (*LMT*) heuristic proposed in [30]. (In Section 6.6, a variety of heuristics are compared by adapting them for a simple common model.)

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 341 of 411

The LMT heuristic first establishes levels of subtasks in the following way. The subtasks that have no input gdis are at the highest level. Each of the remaining subtasks is at one level below the lowest producer of its gdis. The subtasks at the highest level are to be considered first. The LMT heuristic averages the estimated execution times for each subtask across all machines. At each level, a level-average execution time i.e., the average of the machine-average execution times of all subtasks at this level, is also computed. If there are some levels between a subtask and its closest child subtask, the level-average execution time of each middle level is subtracted from the machine-average execution time of this subtask. The adjusted machine-average execution times of the subtasks are used to determine the priorities of the subtasks within each level, i.e., a subtask with a larger average is to be considered earlier at its level. If the number of subtasks at a level is greater than the number of machines in the HC suite, the subtasks with smaller averages are merged so that as a result, the number of combined subtasks at each level equals the number of machines available. When a subtask is being considered, it is assigned to the fastest machine available from those machines that have not yet been assigned any subtasks from the same level.

Another nonevolutionary heuristic, the *baseline* (*BL*), was developed as part of this GA research, and the solution it found was incorporated into the initial population. Similar to the LMT heuristic, the BL heuristic first establishes levels of subtasks based upon their data dependencies. Then all subtasks are ordered such that a subtask at a higher level comes before one at a lower level. The subtasks in the same level are arranged in descending order of their number of output gdis (ties are broken arbitrarily). The subtasks are then scheduled as follows. Let the ith subtask in this order be σ_i, where $0 \leq i < |S|$. First, subtask σ_0 is assigned to a machine that gives the shortest completion time for σ_0. Then, the heuristic evaluates $|M|$ assignments for σ_1, each time assigning σ_1 to a different machine, with the previously decided machine assignment of σ_0 left unchanged. The subtask σ_1 is finally assigned to a machine that gives the shortest overall completion time for both σ_0 and σ_1. The BL heuristic continues to evaluate the remaining subtasks in the order defined earlier. When scheduling subtask σ_i, $|M|$ possible machine assignments are evaluated, each time with the previously decided machine assignments of subtasks σ_j $(0 \leq j < i)$ left unchanged. Subtask σ_i is finally assigned to a machine that gives the shortest overall completion time of subtasks σ_0 through σ_i. The total number of evaluations is thus $|S| \times |M|$, and only i subtasks (out of $|S|$) are considered when performing evaluations for the $|M|$ machine assignments for subtasks σ_i.

To determine the best GA parameters for solving larger HC mapping problems, 50 larger scenarios were randomly generated in each communication category. Each of these scenarios contained 50 subtasks and 5 machines. For each scenario, GA runs were conducted for the following combinations of crossover probability and mutation probability. The crossover probability ranged from 0.1 to 1.0 in steps of 0.1, and the mutation probability ranged from 0.04 to 0.40 in steps of 0.04 and from 0.4 to 1.0 in steps of 0.1. Let the *relative solution quality* be the task completion time of the solution found by the LMT heuristic divided by that found by the approach being investigated. A greater value of the relative solution quality means that the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 342 of 411

approach being investigated finds a better solution to the HC mapping problem (i.e., with a shorter overall completion time for the application task represented by the SPDAG). With each crossover and mutation probability pair and for each communication load, the average relative solution quality of the 50 GA runs, each on a different scenario, was computed.

For each communication load category, a *region of good performance* could be identified for a range of crossover and mutation probabilities. The regions of good performance generally consisted of moderate-to-high crossover probability and low-to-moderate mutation probability. This is consistent with the results from the GA literature, which show that crossover is GA's major operator, and mutation plays a secondary role in GA searches [14, 25, 49].

The crossover and mutation probabilities with which the best relative solution quality had been achieved were used in each corresponding communication load category. When mapping real tasks, the communication load can be determined by computing the ratio of the number of gdis to the number of subtasks. Once the communication load category is known, a probability pair from the corresponding region of good performance can be used.

On Sun Sparc5 workstations, for these larger scenarios, both the LMT heuristic and the BL heuristic took no more than one minute of CPU time to execute. The average CPU execution time of the GA-based approach on these scenarios ranged from less than one minute for the smallest scenarios (i.e., five subtasks, two machines, and light communication load) to about three and one-half hours for the largest scenarios (i.e, 100 subtasks, 20 machines, and heavy communication load). Recall that it is assumed that this GA-based approach will be used for application tasks that are large production jobs such that the one-time investment of this high execution time is justified.

The performance of the GA-based approach was also compared with that of a random search. For each iteration of the random search, a chromosome was randomly generated, this chromosome was evaluated, and the fitness value was compared with the saved best fitness value. If the fitness value of the current chromosome was better than the saved best value, it became the saved best fitness value. For each scenario, the random search iterated for the same length of time as that taken by the GA-based approach on the same scenario.

Figure 6.11 shows the performance comparisons between the LMT heuristic and the GA-based approach for heavily communicating larger scenarios. The GA-based approach used a crossover probability of 1.0 and a mutation probability of 0.2. These values were the best performing probabilities over a wide range of combinations examined for a separate set of test scenarios. In the figure, the horizontal axes are the number of subtasks in log scale. The vertical axes are the relative solution quality of the various approaches. Each point in the figure is the average of 50 independent scenarios. The performance comparisons among the GA-based approach, the LMT heuristic, the BL heuristic, and the random search for moderately communicating and lightly communicating larger scenarios can be found in [58].

In all cases, the GA-based approach presented here outperformed the BL and LMT heuristics and the random search. The improvement of the GA-based approach

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 343 of 411

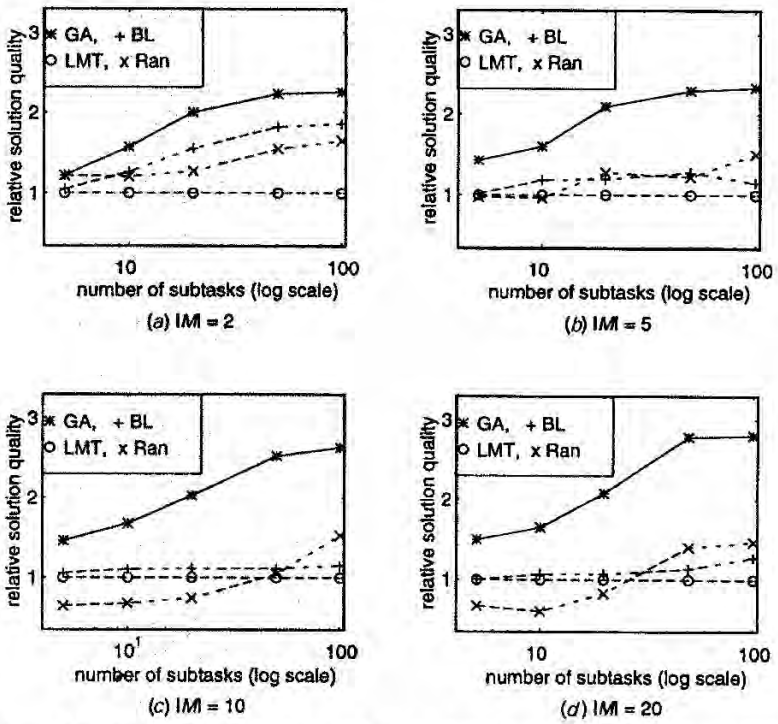


Figure 6.11 Performance comparisons of the GA-based approach with a crossover probability of 1.0 and a mutation probability of 0.2, relative to the LMT heuristic for heavily communicating larger scenarios in (*a*) a two-machine suite, (*b*) a five-machine suite, (*c*) a ten-machine suite, and (*d*) a 20-machine suite. The relative performance of the BL heuristic and the random search are also shown.

over the others showed an overall trend to increase as the number of subtasks increased. The exact shape of the GA-based-approach performance curves is not as significant as the overall trends because the curves are for a heuristic operating on randomly generated data, resulting in some varied performance even when averaged over 50 scenarios for each data point.

6.4.9 Summary

A novel GA-based approach for task mapping in HC environments was presented. This GA-based approach can be used in a variety of HC environments because it does not rely on any specific communication subsystem model. It is applicable to the static scheduling of production jobs, and can be readily used for scheduling multiple

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 344 of 411

independent tasks (and their subtasks) collectively. For small-scale scenarios, the proposed approach found optimal solutions. For larger scenarios, it outperformed two nonevolutionary heuristics and a random search.

There are a number of ways this GA-based approach for HC task mapping can be built upon for future research. These include extending this approach to allow multiple producers for each of the global data items, parallelizing the GA-based approach, developing evaluation procedures for other communication subsystems, and considering loop and data-conditional constructs that involve multiple subtasks.

6.5 SEMISTATIC MATCHING AND SCHEDULING OF SUBTASKS

6.5.1 Overview

In this section, the application domain and HC platform assumed for the semistatic approach are described [7, 8]. This is followed by a discussion of the enhancements made to the GA described in Section 6.4 for determining off-line mappings in this situation. Finally, Section 6.5.7 summarizes the results of an extensive performance study of a simulated semistatic mapping system [35].

6.5.2 Application Domain: Automatic Target Recognition

One type of automatic target recognition (ATR) system takes a stream of images from a group of sensors and produces a scene description [54], tracking the movement of possible targets through the sequence of images. A simplified example of an ATR task can be found in [35]. The various types of image-processing elements required in an ATR system can be broadly classified into three groups: low-level processing (numeric computation), intermediate-level processing (quasi-symbolic computation, e.g., where numeric and symbolic types of operations are used to describe surfaces and shapes of objects in the scene), and high-level processing (symbolic computation, e.g., used to produce the scene description) [1, 4]. Each of these subtasks may allow the use of multiple processors of the same type for efficient execution. Heterogeneous parallel architectures consisting of multiple copies of different types of processors (e.g., SHARC DSP and PowerPC RISC processors [16]) are appropriate computing platforms for efficiently handling computational tasks with such diverse requirements.

The class of ATR applications considered can be modeled as an iterative execution (one execution for each image in the stream) of a set of partially ordered subtasks. Thus, the ATR task can be modeled by a DAG in which the nodes represent subtasks and the edges represent the communications among subtasks. It is assumed that an ATR application in this class is a production job that is executed repeatedly. Therefore, it is worthwhile to invest extra off-line time in preparing an effective mapping of the application onto the hardware platform used to execute it.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 345 of 411

6.5.3 An Intelligent Operating System

A major distinctive architectural feature of the envisioned IOS is the capability for the on-line use of off-line computed mappings [7, 8, 35]. The *ATR Kernel* makes decisions on how a given ATR application task should be accomplished, including determining the partial ordering of subtasks and which algorithms could be used to accomplish each subtask. The *HC Kernel* uses a semistatic method to decide how the partially ordered algorithmic suggestions should be implemented and mapped onto the heterogeneous parallel platform. Furthermore, the HC Kernel interacts with the *Basic Kernel* (the low-level operating system) to initiate the application and monitor its execution. This allows the HC Kernel to decide at the end of each iteration through the application if the subtasks should be remapped onto the hardware platform. Information from the *Algorithm Database* and the *Knowledge Base* is used to support the ATR and HC Kernels.

The Algorithm Database includes one or more implementations of each algorithm (e.g., one for each processor type). The Algorithm Database also contains the expected execution time of each algorithm implementation, typically specified as a function of type and number of processors assigned, interprocessor communication time, and certain input data and application characteristics. Dynamic parameters are those input characteristics, such as amount of clutter, and application characteristics, such as number of located objects to be identified, that (1) will change during run time, (2) can be computed by the application as it executes, and (3) can impact the execution time of each subtask in the task graph. This is an expected time rather than a definite time, because it can vary depending on the actual values of the input data being processed. System parameters, such as the number of each type of processor in the platform, are stored in the Knowledge Base. The off-line and on-line components of the HC Kernel are detailed below. For more details about the IOS, the reader is referred to [8].

6.5.4 Off-line HC Kernel Component: MDDG Generator

A *mapped data-dependency graph* (*MDDG*) is a DAG annotated with the task-to-machine mapping. The *HC Kernel MDDG Generator*, which is a major component of the off-line IOS, is responsible for mapping each DAG onto the heterogeneous parallel hardware platform creating a corresponding MDDG. The structure of the HC Kernel MDDG Generator is such that any effective heuristic could be employed. For the iterative ATR application domain, it is possible to use off-line precomputed mappings to reconfigure resources in real time.

Because different sets of dynamic parameter values can lead to different subtask execution times (e.g., more objects to be recognized in the scene), the HC Kernel MDDG Generator will, in general, generate different mappings for the same DAG (corresponding to different sets of dynamic parameter values). It is assumed that the ranges of the dynamic parameters are known. The space of dynamic parameters is partitioned into a number of disjoint regions, and within each region a random set of representative dynamic parameter vectors are chosen, each of which is called a

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 346 of 411

sample vector. For each sample vector, an off-line mapping heuristic (the enhanced GA described in Section 6.5.6) uses the following information to create a mapping, i.e., to transform a DAG into an MDDG: (1) the structure of the underlying DAG; (2) the expected execution time of each subtask on a set of processors (of the same type) assigned, as a function of the type and number of processors; (3) the intersubtask data transfers needed, in terms of formats and expected sizes of the data items to be transferred; (4) the expected time to send data from one processor to another as a function of the size of the data item to be transferred; and (5) the number of each type of processor that is in the hardware platform.

The mapping for each sample vector is exhaustively evaluated for every other sample vector in the region by applying the mapping to the DAG and computing the task execution time using that other sample vector's dynamic parameter values. The mapping that gives the minimum average execution time is chosen as the representative mapping for the corresponding region in the dynamic parameter space. This representative mapping and the corresponding average execution time are stored in the off-line mapping table (i.e., the *MDDG Table)*, which is a multi-dimensional array, indexed by dynamic parameter ranges. Thus, for a given DAG, the MDDG for each region of the dynamic parameter space is stored in the MDDG table for that DAG, along with the corresponding expected execution time.

6.5.5 On-Line HC Kernel Component: Execution Monitor

The *HC Kernel Monitor* is an on-line component responsible for (1) establishing the initial mapping of the given application onto the hardware platform, and (2) monitoring the execution of the application and, at the end of each iteration through the corresponding DAG, deciding if and how the mapping of the application onto the hardware platform should be changed based on information about the actual values of the dynamic parameters. Examples of dynamic parameters include the contrast level of an image, the number of objects in a scene, and the average size (in pixels) of an object in a scene.

The initial mapping is selected from the MDDG Table for that DAG based on menu-driven input from the application user about an initial value to assume for each of the dynamic parameters. During execution of the application, the HC Kernel Monitor receives actual updated values of the dynamic parameters at the end of each iteration through the corresponding DAG. The HC Kernel Monitor will use the most recent values of these dynamic parameters to estimate if changing the mapping will reduce the expected execution time of the next iteration through the corresponding DAG. It does this by comparing the actual execution time of the last completed iteration with the sum of an estimated time for reconfiguration plus the expected execution time from the MDDG Table entry for the region that includes the most recent dynamic parameter values known. Thus, this decision is made in real time after all subtasks' implementations for the current iteration have finished executing and before any subtask implementations begin to execute for the next iteration. If it is desirable to change the mapping, then the HC Kernel Monitor will pass the

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 347 of 411

MDDG Table entry index to the Basic Kernel for use in the next iteration. If not, the same mapping will continue to be used.

6.5.6 Genetic Algorithm Used for Semistatic Approach

The genetic algorithm described in Section 6.4 is enhanced for determining the off-line mappings. Because each subtask can be mapped to multiple processors of the same type, a new string, called the *allocation string*, is also incorporated into each chromosome in the enhanced GA. Specifically, each entry i in the allocation string represents the number of processors of a certain type (specified by the corresponding entry in the matching string) assigned to the subtask s_i. Typically, multiple subtasks will be assigned to some of the same processors in a processor group. The subtasks are then executed in a nonpreemptive manner based on the ordering that is specified by the scheduling string.

The initial allocation string for each chromosome is generated by randomly selecting a value from 1 to p_{opt} (defined in Section 6.5.7.2) as the number of processors allocated to each subtask (for the processor group type specified in the matching string). The solution generated by a fast heuristic is also included in the initial population. The heuristic used is a fast static scheduling algorithm, called the *earliest completion time* (*ECT*) algorithm [57], which is similar to the LMT heuristic discussed in Section 6.4.8 and is described in [35]. One chromosome with an allocation string where each entry is the optimal number of processors for that subtask (individually) is also included in the initial population.

The mutation operator for the allocation string randomly selects an entry in the string and locally optimizes it by changing the number of processors to a value that gives the best total task execution time. The crossover operator for allocation strings is the same as the one used for matching strings. The probability for mutation and crossover for all strings in the experiments below is 0.4.

The GA is executed 10 times for each sample vector. To enhance diversity, only 5 of the 10 runs include a chromosome generated by the ECT algorithm in the initial population.

6.5.7 Performance Results

6.5.7.1 Overview In [34, 35], an extensive performance study was performed on a simulated HC Kernel implemented using the enhanced GA as the off-line heuristic. The goal of the study was to evaluate the ideas underlying the particular semistatic method of on-line use of off-line-derived mappings described earlier (referred to an *On–Off* in subsequent sections). Four approaches were compared in the experiments: (1) the On–Off approach; (2) the ECT algorithm as a dynamic scheduling algorithm; (3) the infeasible approach of using the GA as a dynamic scheduling algorithm (referred to as *GA On-line*) and (4) an ideal but impossible approach that uses the GA On-line with the exact (as yet unknown) dynamic parameters for the iteration to be executed next (referred to as *Ideal*). In the latter two schemes, a mapping determined for a previous iteration is also included in the initial

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 348 of 411

population of the current iteration, and reconfiguration times are ignored. These two infeasible schemes were merely used as references for comparison to solution quality of the former two approaches.

6.5.7.2 System Model To evaluate the On–Off semistatic mapping methodology, an example architecture was chosen [35]. It consisted of 4 different types of processors with 16 processors for each type (i.e., total number of processors in the HC platform is 64). The processors within each type are connected via a 17-port crossbar switch, whereas the 4 different types of processor are connected using a 4-port crossbar switch. The execution time of a subtask and the communication time between subtasks in an application task were modeled by equations that are functions of the dynamic parameters. Specifically, the simple execution time expression used in this task model is a version of Amdahl's law extended by a term representing the parallelization overhead (e.g., synchronization and communication). The serial and parallel fractions of a subtask are frequently represented using similar models (e.g., [9, 43]). The execution-time expression for subtask s_i includes: (1) three generic dynamic parameters, α, β, and γ; (2) the number of processors used, p; and (3) three coefficients, a_i, b_i, and c_i. The parallel fraction and serial fraction of subtask s_i are represented by $a_i\alpha/p$ and $c_i\gamma$, respectively. The parallelization overhead is represented by $b_i\beta \log p$. The relative speed of a subtask s_i on a processor of type u is given by the heterogeneity factor h_{iu}. The execution time of subtask s_i is then given by the expression: $b_{iu}(a_i\alpha/p + b_i\beta \log p + c_i\gamma)$. By differentiating this equation and equating it to zero, the optimal value of p (p_{opt}) that leads to the minimum execution time for a given subtask is $(a_i\alpha)/(b_i\beta)$.

It is assumed that the size of the data to be transferred between two subtasks s_i and s_j consists of a fixed portion d_{ij} and a variable portion $e_{ij}\mu$, where μ is a fourth dynamic parameter. For communication between virtual machines whose processor types are u and v, s_{uv} and R_{uv} are the message start-up time and the data transmission rate, respectively. The intersubtask communication time c_{uv} between subtask s_i on a virtual machine with processors of type u and subtask s_j on a virtual machine with processors of type v is given by the expression: $S_{uv} + (d_{ij} + e_{ij}\mu)/R_{uv}$.

The example architecture and the simplified generic equations are used as input to the simulated HC Kernel only. The On–Off method can be adopted for other target architectures and the actual time equations specified by the application developer. In practice, a particular single dynamic parameter can impact any subset of the components of a given subtask's execution time equation.

6.5.7.3 Workload To investigate the performance of the On–Off approach, randomly generated DAGs containing 10, 50, 100, or 200 were used. These DAGs included regularly structured graphs (in-trees, out-trees, and fork-joins [34, 35]) and randomly structured DAGs. For each size and structure, 10 different graphs were generated, and thus a total of 160 graphs were used. The input to the simulated on-line module consists of an execution profile that is composed of a certain number of iterations of executing the DAG. Examples of two randomly generated execution profiles containing 20 iterations are shown in Table 6.1. In each profile, the dynamic

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 349 of 411

TABLE 6.1 Execution Profiles of Dynamic Parameters

	Profile A					Profile B			
Iteration	α	β	γ	μ	Iteration	α	β	γ	μ
0	3,000	15	300	60	0	3,000	15	300	60
1	2,821	15	287	63	1	4,309	15	409	82
2	2,949	12	302	65	2	2,635	7	268	43
3	3,073	12	286	68	3	3,894	6	361	27
4	3,228	11	273	71	4	2,241	8	197	39
5	3,090	13	258	67	5	1,265	12	287	52
6	3,256	11	272	70	6	1,699	16	420	75
7	3,424	16	259	73	7	1,138	11	282	50
8	3,621	16	271	75	8	1,543	12	153	67
9	3,811	13	260	78	9	2,205	17	225	97
10	4,014	17	245	81	10	3,198	10	332	51
11	4,229	13	257	77	11	4,678	18	477	73
12	3,994	19	242	80	12	2,588	8	315	48
13	4,179	15	253	83	13	1,358	16	211	67
14	4,386	15	264	78	14	1,794	17	307	98
15	4,208	13	249	82	15	2,605	11	163	61
16	4,016	14	236	77	16	3,719	17	240	87
17	3,835	16	226	81	17	2,478	9	332	53
18	4,026	19	238	84	18	1,507	16	466	76
19	4,258	16	251	88	19	2,081	8	243	50
20	4,479	15	265	92	20	3,053	17	149	70

Notes: Profile A, average percentage change in dynamic parameter values = 5%. Profile B, average percentage change = 40%.

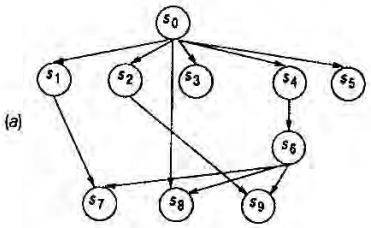
parameter values change from one iteration to another within certain ranges (α: [1,000–5,000], β: [5–25], γ: [100–500], and μ: [20–100]). Specifically, the average percentage change in dynamic parameter values in Profile A and Profile B are 5% and 40%, respectively. In each profile, row i represents the values of the dynamic parameters observed after execution of the DAG for iteration i is finished. Thus, when execution of the task iteration begins, the on-line module does not know the (simulated) actual values of the dynamic parameters for that iteration. The on-line module has to determine a mapping for iteration i based on the dynamic parameter values of iteration $i - 1$. The methods for generating the DAGs and execution profiles can be found in [34, 35].

In generating the off-line mapping tables, each dynamic parameter range is partitioned into four equal intervals, creating $4^4 = 256$ regions. Thus, the MDDG Table stores 256 mappings. Ten sample vectors are randomly chosen from each region. Because the GA for each sample vector is performed with 10 different initial populations, the GA is executed $256 \times 10 \times 10 = 25{,}600$ times to build each MDDG Table. The reconfiguration time is assumed to be 1,000 for these experiments.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 350 of 411

6.5.7.4 Results First consider the results of scheduling a 10-node random task DAG using the two execution profiles. The structure and parameters of an example 10-node random DAG are shown in Fig. 6.12. Detailed results of using the four approaches for Profile A are shown in Table 6.2. The definitions of the data columns are (1) t(map[$i-1$]): the task execution time of iteration i using the mapping chosen at the end of iteration $i-1$, denoted by map[$i-1$]; (2) t'(tab[$i-1$]): the task execution time of the mapping stored in the MDDG Table, denoted by tab[$i-1$], of the sample vector whose region includes the dynamic parameter values at iteration $i-1$; (3) rc: the reconfiguration time, if remapping is performed; (4) t(ect[$i-1$]): the execution time of the mapping, denoted by ect[$i-1$], determined using the ECT algorithm with the parameters at iteration $i-1$; (5) t(ga[$i-1$]): the task execution time of iteration i by applying the mapping determined by the GA using the dynamic parameter values from iteration $i-1$; and (6) t(ga[i]): the task execution time of iteration i determined by the GA using the exact dynamic parameter values for iteration i.

As can be seen from Table 6.2, the On–Off approach of dynamically using off-line-derived mappings resulted in much smaller total execution time (1,115,545) compared to that of using the ECT algorithm (1,668,705). The improvement is



(b)

s_i	a_i	b_i	c_i
s_0	9	24	49
s_1	42	61	9
s_2	42	50	43
s_3	2	34	47
s_4	9	10	38
s_5	33	59	76
s_6	63	45	29
s_7	56	14	54
s_8	48	68	10
s_9	36	25	29

(c)

	d_{ij}	e_{ij}
$s_0 \rightarrow s_1$	5	3
$s_0 \rightarrow s_2$	6	6
$s_0 \rightarrow s_3$	2	1
$s_0 \rightarrow s_4$	7	5
$s_0 \rightarrow s_5$	3	7
$s_0 \rightarrow s_8$	5	6
$s_1 \rightarrow s_7$	3	2
$s_2 \rightarrow s_9$	7	7
$s_4 \rightarrow s_6$	4	2
$s_6 \rightarrow s_7$	5	6
$s_6 \rightarrow s_8$	7	8
$s_6 \rightarrow s_9$	9	10

(d)

	h_{i0}	h_{i1}	h_{i2}	h_{i3}
s_0	0.4897	0.6815	0.7711	0.7503
s_1	0.2926	0.7129	0.4511	0.2725
s_2	0.2575	0.8511	0.5096	0.8779
s_3	0.6337	0.6921	0.8479	0.3451
s_4	0.8283	0.2745	0.4114	0.2838
s_5	0.7267	0.3124	0.2600	0.3354
s_6	0.3932	0.7026	0.8072	0.8066
s_7	0.5276	0.7990	0.5081	0.7942
s_8	0.5876	0.3863	0.6515	0.6472
s_9	0.8760	0.8794	0.6965	0.2407

Figure 6.12 (a) An example of a 10-node randomly generated task graph; (b) coefficients of the subtask execution-time equations; (c) coefficients of the intersubtask communication data equations; (d) heterogeneity factors h_{iu} for the subtask execution-time equation.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 351 of 411

TABLE 6.2 Results for the 10-Node Random Graph using Profile A

	On–Off			ECT			GA On–Line	Ideal
i	t(map[$i-1$])	t'(tab[$i-1$])	rc	t(map[$i-1$])	t(ect[$i-1$])	rc	t(ga[$i-1$])	t(ga[i])
0	—	54,391	1,000	—	74,516	1,000	—	46,228
1	47,508	46,045	1,000	76,384	76,029	0	47,508	47,508
2	47,993	47,286	0	79,227	75,866	1,000	49,095	45,280
3	48,301	51,679	0	75,375	74,586	0	50,306	45,856
4	48,921	51,679	0	76,912	74,755	1,000	48,520	43,611
5	44,035	51,679	0	74,966	71,491	1,000	53,579	40,937
6	49,121	51,679	0	76,226	75,020	1,000	48,410	46,496
7	53,225	52,891	0	79,468	77,336	1,000	51,161	47,580
8	55,303	52,891	1,000	81,944	79,878	1,000	55,447	52,809
9	56,355	51,679	1,000	83,053	82,556	0	52,846	50,539
10	63,288	63,958	0	86,042	87,381	0	55,887	53,733
11	56,631	52,610	1,000	82,874	88,226	0	58,984	51,973
12	58,025	55,142	1,000	87,122	81,860	1,000	63,183	53,733
13	57,410	63,958	0	87,644	87,182	0	57,948	54,311
14	59,774	53,073	1,000	85,449	87,898	0	56,267	52,328
15	58,932	60,754	0	86,152	87,311	0	59,472	55,476
16	55,878	58,723	0	81,710	81,883	0	55,878	55,878
17	58,200	59,774	0	81,434	76,982	1,000	56,981	51,227
18	60,966	63,958	0	88,524	81,581	1,000	56,961	54,483
19	63,071	63,958	0	91,701	84,474	1,000	60,043	59,846
20	65,608	—	—	95,498	93,014	—	58,528	53,842
Total	1,115,545			1,668,705			1,097,004	1,065,040

Note: "Total" below is the total task execution time for 20 iterations.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 352 of 411

approximately 33%. The On–Off approach consistently resulted in performance that was comparable to the infeasible GA On-line scheme (about 2% worse), and was only marginally outperformed by the Ideal (but impossible) method (about 5% worse). Indeed, one very interesting observation is that at some iterations (i.e., iterations 2, 3, 5, 8, 11, 12, 13, and 15), the On–Off approach generated a mapping

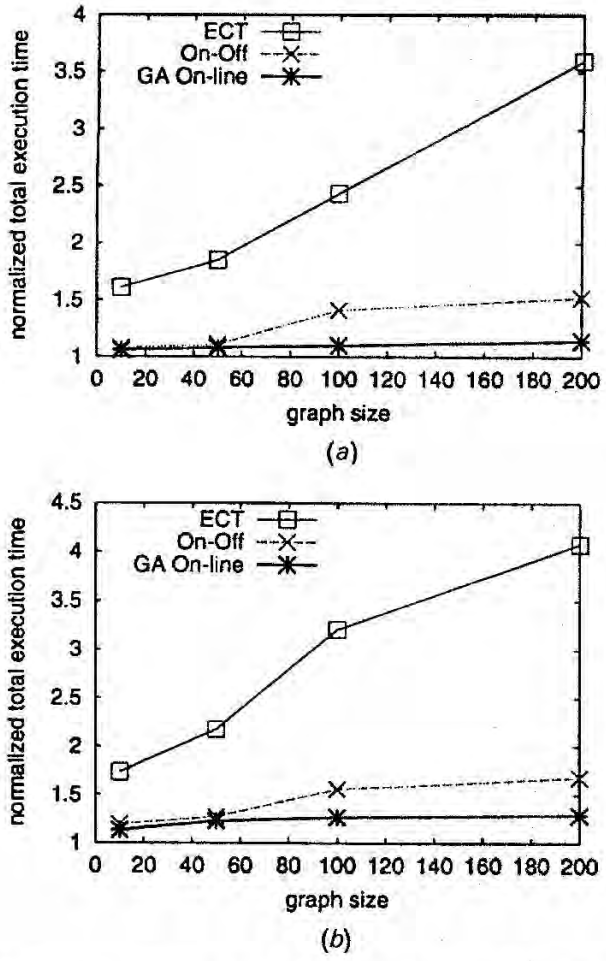


Figure 6.13 Comparison of normalized total task execution times for randomly structured graphs with (*a*) Profile A and (*b*) Profile B.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 353 of 411

resulting in a shorter execution time than the GA On-line approach. Thus, the GA On-line approach, because it computes a mapping optimized for the specific dynamic parameter values from the previous iteration, is sometimes not as robust to changes in the dynamic parameter values as the region-sampling techniques used by the On–Off approach.

Figure 6.13*a* and *b* show the average *normalized total execution times* of the ECT, On–Off, and GA On-line approaches with respect to the Ideal method for the randomly structured graphs. The normalized total execution time of each test case is calculated by dividing the total execution time of a particular approach (e.g., ECT) by that of the Ideal method. Each point on the curves gives the average value of 10 test cases. As can be seen, the normalized total execution times are, in general, slightly higher for Profile B than for Profile A. The normalized total execution times of On–Off and GA On-line are consistently of similar values for all graph sizes. However, the ECT approach performed much worse, especially for large graphs (sizes 100 and 200). An explanation for this phenomenon is because the ECT algorithm employs a strictly greedy scheduling method, the effect of making mistakes at early stages of scheduling can be propagated to later stages. The adverse impact of such a greedy approach can be more profound for larger graphs. For more detailed results, the reader is referred to [34, 35].

6.5.8 Summary

This study focused on the design of a semistatic approach for using GA derived mappings in real time. For the computational environment considered, an HC Kernel was presented for making real-time, on-line, input-data-dependent remappings of the application subtasks to the processors in the heterogeneous parallel hardware platform using previously stored off-line, statically determined mappings. In particular, it was shown how the HC Kernel can be used to create the MDDG Table off-line using a GA with a novel dynamic parameter space partitioning and sampling technique, and then use it to make real-time, on-line decisions and selections of mappings. The simulation results indicated that the semistatic On–Off approach is effective in that it consistently outperformed a fast dynamic mapping heuristic by a considerable margin, and gave reasonable performance even when compared to the impossible approach of using the genetic algorithm on-line with future knowledge of the next iteration's dynamic parameters. The On–Off approach reviewed here can also be used for other application domains and classes of hardware platforms whose characteristics are similar to those of the applications and platforms considered here.

6.6 STATIC MATCHING AND SCHEDULING FOR META-TASKS

6.6.1 Introduction

This study implemented 11 different static meta-task mapping heuristics, including 2 evolutionary approaches, so a comparison could be made of their performance using

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 354 of 411

a common simulated HC environment [6]. Section 6.6.2 presents information about how the ETC matrices were generated. Descriptions of the 11 heuristics implemented appear in Section 6.6.3. Last, a sampling of results from the experiments are shown in Section 6.6.4.

6.6.2. ETC Matrices

For the simulation studies, characteristics of the ETC matrices were varied in an attempt to represent a range of possible HC environments. The ETC matrices used were generated using the following method. Initially, a $|T| \times 1$ *baseline column vector*, B, of floating-point values is created. Let ϕ_b be the upper bound of the range of possible values within the baseline vector. The baseline column vector is generated by repeatedly selecting a uniform random number, $x_b^i \in [1, \phi_b)$, and $B(i) = x_b^i$ for $0 \leq i < |T|$. Next, the rows of the ETC matrix are constructed. Each element ETC(i, j) in row i of the ETC matrix is created by taking the baseline value, $B(i)$, and multiplying it by a uniform random number, $x_r^{i,j}$, which has an upper bound of ϕ_r. This new random number, $x_r^{i,j} \in [1, \phi_r)$, is called a *row multiplier*. One row requires $|M|$ different row multipliers, $0 \leq j < |M|$. Each row i of the ETC matrix then can be described as $\text{ETC}(i, j) = B(i) \times x_r^{i,j}$, for $0 \leq j < |M|$. The baseline column itself does not appear in the final ETC matrix.) This process is repeated for each row until the $|M| \times |T|$ ETC matrix is full. Therefore any given value in the ETC matrix is within the range $[1, \phi_b \times \phi_r)$.

To evaluate the heuristics for different mapping scenarios, the characteristics of the ETC matrix were varied based on several different methods from [3]. The amount of variance among the execution times of tasks in the meta-task for a given machine is defined as *task heterogeneity*. Task heterogeneity was varied by changing the upper bound of the random numbers within the baseline column vector. *High* task heterogeneity was represented by $\phi_b = 3{,}000$, and *low* task heterogeneity by $\phi_b = 100$. *Machine heterogeneity* represents the variation that is possible among the execution times for a given task across all the machines. Machine heterogeneity was varied by changing the upper bound of the random numbers used to multiply the baseline values. *High* machine heterogeneity values were generated using $\phi_r = 1{,}000$, while *low* machine heterogeneity values used $\phi_r = 10$. These heterogeneous ranges are based on one type of expected environment for MSHN. The ranges were chosen to reflect the fact that in real situations there is more variability across task execution times on a given machine than the execution time for a single task across different machines.

To further vary the ETC matrix in an attempt to capture more aspects of realistic mapping situations, different ETC matrix consistencies were used. An ETC matrix is said to be *consistent* if whenever a machine j executes any task i faster than machine k, then machine j executes all tasks faster than machine k [3]. Consistent matrices were generated by sorting each row of the ETC matrix independently. In contrast, *inconsistent* matrices characterize the situation where machine j is faster than machine k for some tasks, and slower for others. These matrices are left in the unordered, random state in which they were generated. *Semiconsistent* matrices are

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 355 of 411

TABLE 6.3 Sample 8 × 8 Excerpt from an ETC Matrix with Inconsistent, High-Task, High-Machine Heterogeniety

	Machines							
Tasks	436,735	815,309	891,469	1,722,197	1,340,988	740,028	1,749,673	251,140
	950,470	933,830	2,156,144	2,202,018	2,286,210	2,779,669	220,536	1,769,184
	453,126	479,091	150,324	386,338	401,682	218,826	242,699	11,392
	1,289,078	1,400,308	2,378,363	2,458,087	351,387	925,070	2,097,914	1,206,158
	646,129	576,144	1,475,908	424,448	576,238	223,453	256,804	88,737
	1,061,682	43,439	1,355,855	1,736,937	1,624,942	2,070,705	1,977,650	1,066,470
	10,783	7,453	3,454	23,720	29,817	1,143	44,249	5,039
	1,940,704	1,682,338	1,978,545	788,342	1,192,052	1,922,914	701,336	1,052,728

inconsistent matrices that include a consistent submatrix. For the semiconsistent matrices used here, the row elements in column positions $\{0, 2, 4, \ldots\}$ of row i are extracted sorted, and replaced in order, while the row elements in column positions $\{1, 3, 5, \ldots\}$ remain unordered. (That is, the even columns are consistent and the odd columns are, in general, inconsistent.)

Sample ETC matrices are shown in Tables 6.3 and 6.4. All results in this study used ETC matrices that were of size $|T| = 512$ tasks by $|M| = 16$ machines. While it was necessary to select some specific parameter values to allow implementation of a simulation, the characteristics and techniques presented here are completely general. Therefore, if these parameter values do not apply to a specific situation of interest, researchers may use other ranges, distributions, matrix sizes, etc.

4.6.3 Description of Heuristics

The definitions of the 11 static meta-task mapping heuristics are provided below. First, some preliminary terms must be defined. *Machine availability time*, avail(j), is the earliest time a machine j can complete the execution of all the tasks that previously have been assigned to it. The *completion time* (ct) for a new task i on machine j is ct(i, j) or the machine availability time plus the execution time of task i

TABLE 6.4 Sample 8 × 8 Excerpt from an ETC Matrix with Inconsistent, Low-Task, Low-Machine Heterogeneity

	Machines							
Tasks	512	268	924	494	611	606	921	209
	8	16	23	19	27	22	19	8
	228	238	107	180	334	88	192	125
	345	642	136	206	559	349	640	664
	117	235	149	71	136	363	182	359
	240	412	259	319	237	338	178	537
	462	93	574	449	421	559	487	298
	119	36	224	194	176	156	182	192

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 356 of 411

on machine j, i.e., $\text{ct}(i, j) = \text{avail}(j) + \text{ETC}(i, j)$. The performance criterion used to compare the results of the heuristics is the maximum value of $\text{ct}(i, j)$, for $0 \leq i < |T|$ and $0 \leq j < |M|$, for each heuristic, also known as the *makespan* [39]. Each heuristic is attempting to minimize the makespan (i.e., finish execution of the meta-task as soon as possible).

The descriptions below implicitly assume that the machine availability times are updated after each task is mapped. For cases when tasks can be considered in an arbitrary order, the order in which the tasks appeared in the ETC matrix was used. Some of the heuristics listed below had to be modified from their original implementation to better handle the environment under consideration.

For many of the heuristics, there are control parameter values and/or control-function specifications that can be selected for a given implementation. For the studies here, such values and specifications were selected based on experimentation and/or information in the literature. A more thorough description of each of the heuristics and some of the intuition behind them, along with a listing of some alternative implementations, can be found in [6].

OLB *Opportunistic load balancing* (*OLB*) assigns each task, in arbitrary order, to the next available machine, regardless of the task's expected execution time on that machine [2, 21, 22].

UDA In contrast to OLB, *user-directed assignment* (*UDA*) assigns each task, in arbitrary order, to the machine with the best expected execution time for that task, regardless of that machine's availability. UDA is sometimes referred to as limited best assignment (LBA), as in [2, 21]. In general, this heuristic is obviously not applicable to HC environments characterized by consistent ETC matrices.

Fast Greedy *Fast Greedy* assigns each task, in arbitrary order, to the machine with the minimum completion time for that task [2].

Min–min The *Min–min* heuristic begins with the set U of all unmapped tasks. Then, the set of *minimum completion times*, $\text{MCT} = \{\text{mct}_i : \text{mct}_i = \min_{0 \leq j < |M|}(\text{ct}(i, j))$, for each $i \in U\}$, is found. Next, the task from U with the overall *minimum* completion time is selected and assigned to the corresponding machine (hence the name Min–min). Last, the newly mapped task is removed from U, and the process repeats until tasks are mapped (i.e., U is empty) [2, 21, 29].

Max–min The *Max–min* heuristic is very similar to Min–min. The Max–min heuristic also begins with the set U of all unmapped tasks. Then, the set of MCTs is found. Next, the task from U with the overall *maximum* completion time is selected and assigned to the corresponding machine (hence the name Max–min). Last, the newly mapped task is removed from U, and the process repeats until all tasks are mapped (i.e., U is empty) [2, 21, 29].

Greedy The *Greedy* heuristic is literally a combination of the Min–min and Max–min heuristics. The Greedy heuristic performs both of the Min–min and Max–min heuristics, and uses the better solution [2, 21].

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 357 of 411

GA The *genetic algorithm* (*GA*) implemented in this study was adapted from [58] (see Section 6.4 for a description of [58]) to be applied to meta-tasks, and unless otherwise noted, uses similar values and techniques. The GA operates on a population of 200 chromosomes for a given meta-task. Each chromosome is a $|T| \times 1$ vector, where position i ($0 \leq i < |T|$) represents task i, and the entry in position i is the machine to which the task has been mapped. The makespan is the fitness value. The initial population is generated using two methods: (1) 200 randomly generated chromosomes from a uniform distribution, or (2) one chromosome (seed) that is the Min–min solution and 199 random solutions (mappings). The probability of crossover was 60% and mutation was 40%. The stopping criteria that usually occurred in testing were "no changes in the elite chromosome in 150 iterations". Eight GA runs were performed (four times with different initial populations from each method), and the best of the eight mappings is used as the final solution

SA *Simulated annealing* (*SA*) is an iterative technique that considers only one possible solution (mapping) for each meta-task at a time. This technique uses the same representation for a solution as the chromosome for the GA. Mutations of the current chromosome are also performed similarly to the GA.

The corresponding selection process for SA uses a procedure that probabilistically allows poorer solutions to be accepted to attempt to obtain a better search of the solution space (e.g., [13, 33, 42]). This probability is based on a *system temperature* that decreases for each iteration. As the system temperature "cools," it is more difficult for currently poorer solutions to be accepted. The initial system temperature is the makespan of the initial (random) mapping.

Therefore, the SA begins with the current chromosome, mutates it, and then uses the system temperature to determine whether to accept or reject this new solution. After each mutation, the system temperature is decreased by 10%. This represents one iteration of SA. The heuristic stops when there is no change in the makespan of the solution for 150 iterations or the system temperature reaches zero.

GSA The *genetic simulated annealing* (*GSA*) heuristic is a combination of the GA and SA techniques [10, 45]. In general, GSA follows procedures similar to the GA just outlined. GSA operates on a population of 200 chromosomes, uses a Min–min seed in four out of eight initial populations, and performs similar mutation and crossover operations. However, for the selection process, GSA uses the SA cooling schedule and system temperature, and a simplified SA decision process for accepting or rejecting new chromosomes. GSA also employs elitism to guarantee that the best solution always remains in the population.

Tabu *Tabu* search is a solution space search that keeps track of the regions of the solution space that have already been searched, so as not to repeat a search near these areas [15, 24]. A solution (mapping) uses the same representation as a chromosome in the GA approach.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 358 of 411

The implementation of Tabu search used here begins with a random mapping, generated from a uniform distribution. Starting with the first task in the mapping, task $i = 0$, each possible pair of tasks is formed, (i, j) for $0 \leq i < |T| - 1$ and $i < j < |T|$. As each pair of tasks is formed, they potentially exchange machine assignments. This constitutes a *short hop*. After each exchange, the new makespan is evaluated. If the new makespan is an improvement, the new exchange is retained, forming a new mapping (a *successful short hop*). New short hops are generated until a maximum number of successful hops have been made (see next paragraph) or all combinations of task pairs have been exhausted with no further improvement.

At this point, the final mapping from the local solution space search is added to the *tabu list*. Next, a new random mapping is generated, and it must differ from each mapping in the tabu list by at least half of the machine assignments (a *successful long hop*). Then the short hops are repeated. The final stopping criterion for the heuristic is a total of 1200 successful hops (short and long combined). Then, the best mapping from the tabu list is the final answer.

*A** *A** has been applied to many other task-allocation problems (e.g., [11, 31, 42, 44]). The technique used here is similar to [11].

A* is a tree search beginning at a root node that is a null solution. As the tree grows, intermediate nodes represent partial solutions (a subset of tasks are assigned to machines). The partial solution of a child node has one more task mapped than the parent node. Call this additional task a. Each parent node generates $|M|$ children, one for each possible mapping of a. Based on experimentation and a desire to keep execution time of the heuristic tractable the maximum number of leaf nodes in the tree at any one time is limited in this study to $n_{\max} = 1{,}024$.

Each node, n, has a *cost function*, $f(n)$, associated with it. The cost function is an estimated lower bound on the makespan of the best solution, which includes the partial solution represented by node n. (The lower bound on the time for executing the remaining tasks includes the assumptions that each task is assigned to its preferred machine and that all machines are equally utilized.)

Thus, beginning with the root, the node with the minimum $f(n)$ is expanded by its $|M|$ children, until $n_{\max}$ leaf nodes are created. From that point on, any time a node is added, the tree is pruned by deleting the leaf node with the largest $f(n)$. This process continues until a leaf node representing a complete mapping is reached. Note that if the tree is not pruned, this method is equivalent to an exhaustive search.

6.6.4 Results

An interactive software tool has been developed that allows simulation, testing, and demonstration of the heuristics examined in Section 6.6.3. This software tool operates on the meta-tasks defined by the ETC matrices described in Section 6.6.2.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 359 of 411

The software allows a user to specify $|T|$ and $|M|$, to select which types of ETC matrices to use, and to choose which heuristics to execute. It then generates the specified ETC matrices, executes the desired heuristics, and displays the results, similar to Fig. 6.14. The results discussed in this section were generated using portions of this software.

When comparing mapping heuristics, the execution time of the heuristics themselves is an important consideration. For the heuristics listed, the execution times varied greatly. The experimental results discussed below were obtained on a Pentium II 400-MHz processor with 1 GB of RAM. Each of the simpler heuristics [OLB, UDA, Fast Greedy, and Greedy (which includes both Min–min and Max–min)] executed in a few seconds for one ETC matrix with $|T| = 512$ and $|M| = 16$. For the same-sized ETC matrix, SA and Tabu, both of which manipulate a single solution during an iteration, averaged less than 30 s. GA and GSA required approximately 60 s per matrix because they manipulate entire populations, and A* required about 20 min per matrix.

The resulting meta-task execution times (makespans) from the simulations of sample HC environments are shown in Fig. 6.14. All experimental results represent the execution time of a meta-task (defined by a particular ETC matrix) based on the mapping found by the heuristic specified, averaged over 100 different ETC matrices of the same type (i.e., 100 mappings). For each heuristic, the range bars show the minimum and maximum meta-task execution times over the 100 mappings (100 ETC matrices) used to compute the average meta-task execution time.

For the four consistent cases (i.e., each combination of high and low task and machine heterogeneity), the UDA algorithm mapped all tasks to the same machine, resulting in the worst performance by an order of magnitude (therefore, UDA is not included in Fig. 6.14*a*). GA performed the best for the consistent cases. This was due in large part to the good performance of the Min–min heuristic. The best GA solution always came from one of the populations that had been seeded with the Min–min solution. As is apparent in the figure, Min–min performed very well on its own, giving the second best results. However, the mutation, crossover, and selection operations of the GA were always able to improve on this solution. GSA, which also used a Min–min seed, did not always improve upon the Min–min solution. Because of the probabilistic procedure used during selection, GSA would sometimes accept poorer intermediate solutions. These poorer intermediate solutions never led to better final solutions; thus, GSA gave poorer results than the GA. The performance of A* was hindered because the estimates made by $f(n)$ are not as accurate for consistent cases as they are for inconsistent and semiconsistent cases.

These results suggest that if the best overall solution is desired, the GA should be employed. However, the improvement of the GA solution over the Min–min solution was never more than 10%. Therefore, the Min–min heuristic may be more appropriate in certain situations, given the difference in execution times of the two heuristics.

For the four inconsistent test cases, UDA performs very well while the performance of OLB degrades. The OLB performance degradation for the inconsistent

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 360 of 411

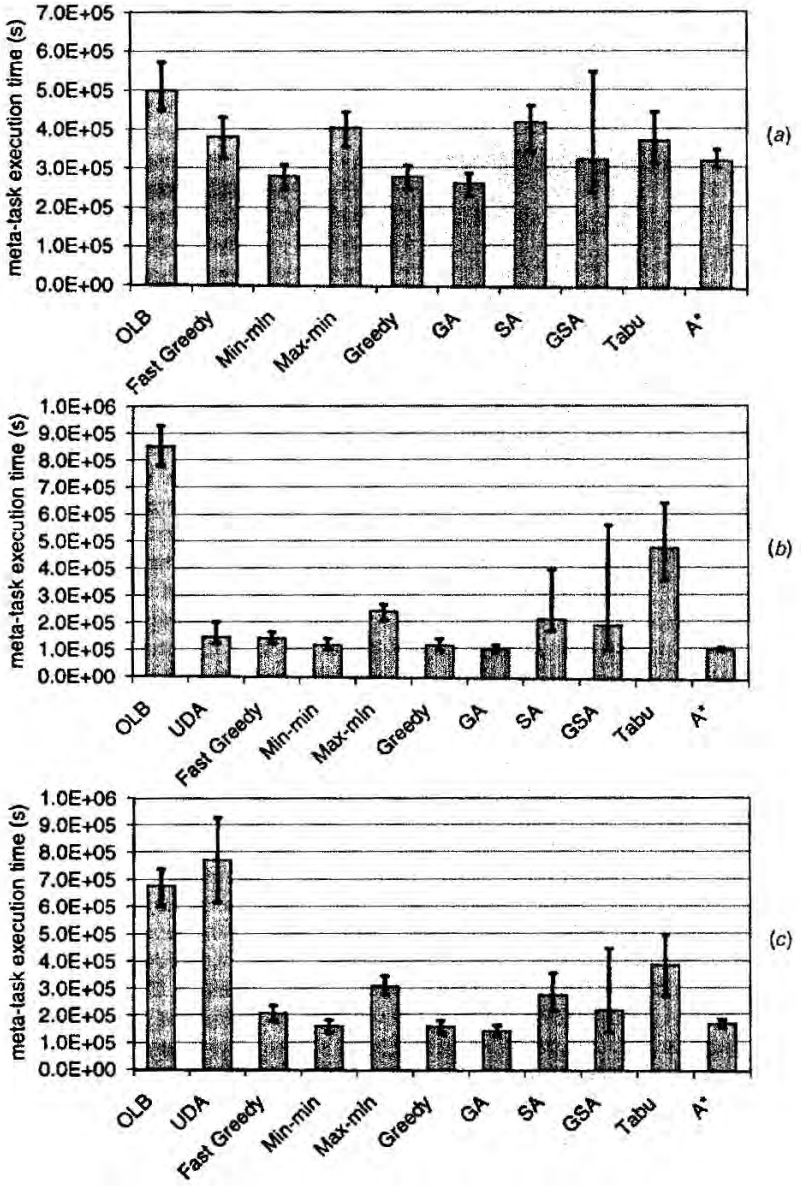


Figure 6.14 Meta-task mapping results for 512 low-heterogeneity tasks and 16 high-heterogeneity machines for ETCs that are (*a*) consistent, (*b*) inconsistent, and (*c*) semiconsistent. The graphs show the mean and range over 100 trials.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 361 of 411

cases can likely be attributed to more "unfavorable" assignments occurring as compared to the consistent cases. For example, for the consistent cases, one machine executes all of the tasks quickest, so more tasks will be assigned to this machine, which is again the best assignment for that task. This phenomenon is less likely to occur for the inconsistent cases because there is no one "best" machine for all of the tasks. In contrast, UDA improves because the "best" machines are distributed across the set of machines, thus task assignments will be more evenly distributed among the set of machines, avoiding load imbalance (to some extent, this is due to the random distributions used in the simulation). Similarly, Fast Greedy and Min–min performed very well, and slightly outperformed UDA, because the machines providing the best task completion times are more evenly distributed among the set of machines. Min–min was also better than Max–min for all of the inconsistent cases. The advantages Min–min gains by mapping "best-case" tasks first outweighs the possible savings in "packing" that Max–min has by mapping "worst-case" tasks first [6].

Tabu gave the second poorest results for the inconsistent cases, with makespans that were always at least 20% worse than Max–min (the third poorest heuristic in terms of mean performance). Inconsistent matrices generated more successful short hops than the associated consistent matrices. Therefore, fewer long hops were generated and less of the solution space was searched, resulting in poorer solutions.

GA and A* had the best average makespans, and were usually within a small constant factor of each other. GA again benefited from having the initial Min–min mapping. A* did well because, if the tasks get more evenly distributed among the machines, this more closely matches the lower-bound estimates $f(n)$.

For semiconsistent cases with high machine heterogeneity, the UDA heuristic again gave the worst results. Intuitively, UDA is suffering from the same problem as in the consistent cases: at least half of all tasks are getting assigned to the same machine. OLB does poorly for high machine heterogeneity cases because worst-case matchings will have higher execution times for high machine heterogeneity. For low machine heterogeneity, the worst-case matchings have a much lower penalty. The best heuristics for the semiconsistent cases were Min–min and GA. This is not surprising because these were two of the best heuristics from the consistent and inconsistent tests, and semiconsistent matrices are a combination of consistent and inconsistent matrices. Min–min was able to do well because it searched the entire row for each task and assigned a high percentage of tasks to their first-choice machine. GA was robust enough to handle the consistent components of the matrices, and did well for the same reason mentioned for inconsistent matrices.

6.6.5 Summary

The goal of this study was to provide insights and a basis for comparison of 11 different heuristics for the mapping of static meta-tasks in different HC environments. The characteristics of the ETC matrices used as input for the heuristics and the methods used to generate them were specified. The implementation of a collection of 11 heuristics from the literature was described. The results of the mapping heuristics were discussed, revealing the best heuristics to use in certain

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 362 of 411

environments. For the situations, implementations, and parameter values used here, GA was the best heuristic for most cases, followed closely by Min–min, with A* also doing well for inconsistent matrices. The comparisons in this study can be used by researchers as a baseline for evaluating the efficacy of new techniques.

6.7 SUMMARY

In a mixed-machine, distributed, heterogeneous computing environment, there is a suite of high-performance machines with different computational capabilities. These machines are interconnected by high-speed links. Such a suite of machines can be used to execute a single application, whose subtasks have diverse execution requirements, or to execute a meta-task, which is a collection of independent tasks with different computational needs.

For the single-application case, subtasks are assigned to and executed on the machines that will result in a minimal execution time for the overall task, considering subtask computation time and intermachine communication overhead. A GA-based approach was described for matching subtasks to machines and scheduling the execution of the subtasks. This approach is used off-line and is based on expected computation and communication times for the subtasks. A dynamic-parameter-sampling-based method for using this approach on-line in certain application domains was also presented. This on-line adaptation may be helpful when computation and communication times can vary significantly from the expected values, depending on the input data being processed.

For the meta-task case, the goal is to match each task to a machine in the suite so that the execution time for the entire meta-task is minimized, In the situation considered here, the matching process occurs off-line and plans the machine assignments and execution schedule for a meta-task for a later time interval (e.g., a large set of production jobs that will execute the next day). In this situation, the matching process can also be used to determine if a proposed set of machines can perform the meta-task within some time limit. The use of GA-based approach for this environment was discussed.

In summary, much work has been done using GAs of various types to solve the problem of matching and scheduling of tasks and meta-tasks in a mixed-machine, distributed, heterogeneous computing environment. This chapter has discussed three types of genetic algorithms that have been studied to solve this problem. Each implementation has particular specifications and qualifications that were being met. In all cases, the genetic algorithm proved to be a useful method for solving the making and scheduling problem being researched.

ACKNOWLEDGMENTS

This research was supported by the DARPA/ITO Quorum Program through GSA subcontract number GS09K99BH0250, an AFCEA Fellowship, the Hong Kong Research Grants Council

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 363 of 411

under contract numbers HKU7124/99E and HKUST6076/97E, and a research initiation grant from the HKU CRCG.

The authors thank N. Beck and A. Naik for their useful comments. We also acknowledge the contributions of our coauthors of the papers [6, 8, 35, 58], which form the basis for this chapter: I. Ahmad, N. Beck, L.L. Bölöni, J.R. Budenske, R.F. Freund, A. Ghafoor, D. Hensgen, M. Maheswaran, R.S. Ramanujan, A.I. Reuther, J.P. Robertson, R. Roychowdhury, L. Wang, and B. Yao.

REFERENCES

1. H. M. Alnuweiri and V. K. Prasanna, Parallel architectures and algorithms for image component labeling. *IEEE Trans. Pattern Anal. Machine Intelligence*, 14(10):1014–1034, Oct. 1992.
2. R. Armstrong, D. Heusgen, and T. Kidd, The relative performance of various mapping algorithms is independent of sizable variances in run-time predictions. In *Proceedings of the Seventh IEEE Heterogeneous Computing Workshop (HCW '98)*, pp. 79–87, Mar. 1998.
3. R. Armstrong, *Investigation of Effect of Different Run-Time Distributions on SmartNet Performance.* Thesis, Department of Computer Science, Naval Postgraduate School Monterey, CA, Sept. 1997.
4. P. Baglietto, M. Maresca, M. Migliardi, and N. Zingirian, Image processing on high-performance RISC systems. *Proc. IEEE*, 84(7):917–930, July 1996.
5. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, A taxonomy for describing matching and scheduling heuristics for mixed-machine heterogeneous computing systems. *Proceedings of the Seventh IEEE Symposium on Reliable Distributed Systems*, 1998, pp. 330–335, Oct. 1998.
6. T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing system. In *Proceedings of the Eighth IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, pp. 15–29, Apr. 1999.
7. J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, Modeling ATR applications for intelligent execution upon a heterogeneous computing platform. In *Proceedings of 1997 IEEE International Conference on Systems, Man and Cybernetics*, pp. 649–656, Oct. 1997.
8. J. R. Budenske, R. S. Ramanujan, and H. J. Siegel, A method for the on-line use of off-line derived remappings of iterative automatic target recognition tasks onto a particular class of heterogeneous parallel platforms. *Jour. Supercomputing*, 12(4):387–406, Oct. 1998.
9. E. A. Carmona and M. D. Rice, Modeling the serial and parallel fractions of a parallel program. *Parallel and Distributed Computing*, 13(3):286–298, Nov. 1991.
10. H. Chen, N. S. Flann, and D. W. Watson, Parallel genetic simulated annealing: A massively parallel SIMD approach, *IEEE Trans. Parallel and Distributed Computing*, 9(2):126–136, Feb. 1998.
11. K. Chow and B. Liu, On mapping signal processing algorithms to a heterogeneous multiprocessor system. In *1991 International Conference on Acoustics, Speech, and Signal Processing (ICASSP '91)*, Vol. 3, pp. 1585–1588, May 1991.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 364 of 411

12. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge University Press, Cambridge, MA, 1992.
13. M. Coli and P. Palazzari, Real time pipelined system design through simulated annealing. *Jour. Sys. Architecture*, 42(6–7):465–475, Dec. 1996.
14. L. Davis, ed., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.
15. I. De Falco, R. Del Balio, E. Tarantino, and R. Vaccaro, Improving search by incorporating evolution principles in parallel tabu search. In *1994 IEEE Conference on Evolutionary Computation*, Vol. 2, pp. 823–828, 1994.
16. T. H. Einstein, Mercury computer systems' modular heterogeneous RACE multicomputer. In *Proceedings of the Sixth IEEE Heterogeneous Computing Workshop (HCW '97)*, pp. 60–71, April 1997.
17. M. M. Eshaghian, ed., *Heterogeneous Computing*. Artech House, Northwood, MA, 1996.
18. M. M. Eshaghian and M. E Shaaban, Cluster-M programming paradigm. *Int. Jour. High Speed Computing*, 6(2):287–309, June 1994.
19. D. Fernandez-Baca, Allocating modules to processors in a distributed system. *IEEE Trans. Software Eng.* SE-15(11):1427–1436, Nov. 1989.
20. R. F. Freund, Optimal selection theory for superconcurrency. *Supercomputing '89*, pp. 699–703, Nov. 1989.
21. R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, Scheduling resources in multi-user heterogeneous, computing environments with SmartNet. In *Proceedings of the Seventh IEEE Heterogeneous Computing Workshop (HCW '98)*, pp. 184–199, March 1998.
22. R. F. Freund and H. J. Siegel, Heterogeneous processing, *IEEE Comuputer*, 26(6):13–17, June 1993.
23. A. Ghafoor and J. Yang, Distributed heterogeneous supercomputing management system. *IEEE Computer*, 26(6):78–86, June 1993.
24. F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, MA, 1997.
25. D. B. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, MA, 1989.
26. D. A. Hensgen, T. Kidd, M. C. Schnaidt, D. St. John, H. J. Siegel, T. D. Braun, M. Maheswaran, S. Ali, J.-K. Kim, C. Irvine, T. Levin, R. Wright, R. F. Freund, M. Godfrey, A. Duman, P. Carff, S. Kidd, V. Prasanna, P. Bhat, and A. Alhusaini, An overview of MSHN: A management system for heterogeneous networks. In *Proceedings of the Eighth IEEE Workshop on Heterogeneous Computing Systems (HCW '99)*, pp. 184–198, April 1999.
27. J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.
28. R. Hoebelheinrich and R. Thomsen, Multiple crossbar network integrated supercomputing framework, *Supercomputing '89*, pp. 713–720, Nov. 1989.
29. O. H. Ibarra and C. E. Kim, Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Jour. ACM*, 24(2):280–289, April 1977.
30. M. A. Iverson, F. Ozguner, and G. J. Follen, Parallelizing existing applications in a distributed heterogeneous environment. In *Proceedings of the Fifth IEEE Heterogeneous Computing Workshop (HCW '95)*, pp. 93–100, April 1995.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 365 of 411

31. M. Kafil and I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency*, 6(3):42–51, July–Sept. 1998.

32. A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, Heterogeneous computing: Challenges and opportunities. *IEEE Computer*, 26(6):18–27, June 1993.

33. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

34. Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, A. Ghafoor, and I. Ahmad, Evaluation of a semi-static approach to mapping dynamic iterative tasks onto heterogeneous computing systems. In *Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99)*, pp. 204–209, June 1999.

35. Y.-K. Kwok, A. A. Maciejewski, H. J. Siegel, A. Ghafoor, and I. Ahmad, *Implementation and Performance Study of a Semi-static Approach to Mapping Dynamic Iterative Tasks onto Heterogeneous Computing Systems*. Technial report HKUST-CS99-15, Department of Computer Science, The Hong Kong University of Science and Technology, in preparation.

36. M. Maheswaran, T. D. Braun, and H. J. Siegel, Heterogeneous distributed computing. In J. Webster, ed., *Encyclopedia of Electrical Engineering and Electronics*, Vol. 8, pp. 679–690, Wiley, New York, 2000.

37. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Jour. Parallel and Distributed Computing*, 59(2):107–121, Nov. 1999.

38. B. Narahari, A. Youssef, and H. A. Choi, Matching and scheduling in a generalized optimal selection theory. In *Proceedings of the Third IEEE Heterogeneous Compuing Workshop (HCW '94)*, pp. 3–8, April 1994.

39. M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

40. J. L. Ribeiro Filho and P. C. Treleaven, Genetic-algorithm programming environments. *IEEE Computer*, 27(6):28–43, June 1994.

41. G. Rudolph, Convergence analysis of canonical genetic algorithms. *IEEE Trans. Neural Networks*, 5(1):96–101, Jan. 1994.

42. S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1995.

43. K. C. Sevcik, Characterizations of parallelism in applications and their use in scheduling. *Performance Evaluation Rev.*, 17(1):171–180, May 1989.

44. C.-C Shen and W.-H. Tsai, A graph matching approach to optimal task assignment in distributed computing system using a minmax criterion. *IEEE Trans. Computers*, C-34(3):197–203, March 1985.

45. P. Shroff, D. W. Watson, N. S. Flann, and R. F. Freund, Genetic simulated annealing for scheduling data-dependent tasks in heterogeneous environments. In *Proceedings of the Fifth IEEE Heterogeneous Computing Workshop (HCW '96)*, pp. 98–104, April 1996.

46. H. J. Siegel, J. K. Antonio, R. C. Metzger, M. Tan, and L. A. Li, Heterogeneous computing. In A. Y. Zomaya ed., *Parallel and Distributed Computing Handbook*, pp. 725–761, McGraw-Hill, New York, 1996.

47. H. J. Siegel, H. G. Dietz and J. K. Antonio, Software support for heterogeneous computing. In A. B. Tucker Jr. Ed., *The Computer Science and Engineering Handbook*, pp. 1886–1909, CRC Press, Boca Raton, FL, 1997.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 366 of 411

48. H. Singh and A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms. In *Proceedings of the Fifth IEEE Heterogeneous Computing Workshop (HCW '96)*, pp. 86–97, April 1996.

49. M. Srinivas and L. M. Patnaik, Genetic algorithms: A survey. *IEEE Computer*, 27(6):17–26, June 1994.

50. V. S. Sunderam, Design issues in heterogeneous network computing. In *IEEE Workshop on Heterogeneous Processing (rev. Ed.)*, pp. 101–112, March 1992.

51. M. Tan, H. J. Siegel, J. K. Antonio, and Y. A. Li, Minimizing the application execution time through scheduling of subtasks and communication traffic in a heterogeneous computing system. *IEEE Trans. Parallel and Distributed Sys.*, 8(8):857–871, Aug. 1997.

52. Y. G. Tirat-Gefen and A. C. Parker, MEGA An approach to system-level design of application-specific heterogeneous multiprocessors. In *Proceedings of the Fourth IEEE Heterogeneous Computing Workshop (HCW '96)*, pp. 105–117, April 1996.

53. D. Tolmie and J. Renwick, HiPPI: Simplicity yields success. *IEEE Network*, 7(1):28–32, Jan. 1993.

54. J. G. Verly and R. I. Delanoy, Model-based automatic target recognition (ATR) system for forwardlooking groundbased and airborne imaging laser radars (LADAR). *Proc. IEEE*, 84(2): 126–163, Feb. 1996.

55. D. W. Watson, J. K. Antonio, H. J. Siegel, and M. J. Atallah, Static program decomposition among machines in an SIMD/SPMD heterogeneous environment with non-constant mode switching cost. In *Proceedings of the Third IEEE Heterogeneous Computing Workshop (HCW '94)*, pp. 58–65, April 1994.

56. D. W. Watson, J. K. Antonio, H. J. Siegel, R. Gupta, and M. J. Atallah, Static matching of ordered program segments to dedicated machines in a heterogeneous computing environment. In *Proceedings of the Fifth IEEE Heterogeneous Computing Workshop (HCW '96)*, pp. 24–37, April 1996.

57. Q. Wang and K. H. Cheng, List scheduling of parallel tasks. *Information Processing Lett.*, 37(5):291–297, March 1991.

58. L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Jour. Parallel and Distributed Computing*, 47(1):1–15, Nov. 1997.

59. C. C. Weems, G. E. Weaver, and S. G. Dropsho, Linguistic support for heterogeneous parallel processing: A survey and an approach. In *Proceedings of the Third IEEE Heterogeneous Computing Workshop (HCW '94)*, pp. 81–88, April 1994.

60. D. Whitley, The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *1989 International Conference on Genetic Algorithms*, pp. 116–121, Morgan Kaufmann, June 1989.

61. A. Zomaya, C. Ward, and B. Macey, Genetic scheduling for parallel processor systems: Comparative studies and performance issues. *IEEE Trans. Parallel and Distributed Sys.*, 10(8):795–812, Aug. 1999.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 367 of 411

Efficient Data-parallel Distributed DNN Training for Big Dataset under Heterogeneous GPU Cluster

Shinyoung Ahn, Sookwang Lee*
Supercomputing System Research Section
ETRI
Daejeon, Republic of Korea
{syahn, 2sookwang}@etri.re.kr

Hyeonseong Choi
Software Team
MangoBoost
Seoul, Republic of Korea
chyon794@gmail.com

Jaehyun Lee
Research Institute
Puzzle AI Inc.
Daejeon, Republic of Korea
jay@puzzle-ai.com

2024 IEEE International Conference on Big Data (BigData) | 979-8-3503-6248-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/BigData62323.2024.10825722

***Abstract*—Training large-scale deep neural networks (DNNs) using a large number of parameters requires significant computational resources. Despite the rapid advancements in GPU technology, limited budgets have forced many institutions to gradually build GPU servers, leading to growing challenges with resource heterogeneity. However, most open-source distributed deep-learning libraries use synchronous training algorithms that perform better on homogeneous GPUs than on heterogeneous GPUs. Therefore, many researchers have struggled to efficiently conduct distributed training on heterogeneous GPU clusters owing to the straggler problem. In this study, we introduce Efficient Distributed Deep learning IIbrary based on SoftMemoryBox(EDDIS), a novel data-parallel distributed deep learning library. EDDIS overcomes the scalability limitations caused by heterogeneity, enabling the efficient utilization of heterogeneous GPU resources. EDDIS trains DNNs synchronously, asynchronously, and in a hybrid manner and supports TensorFlow and PyTorch. In a heterogeneous GPU environment, EDDIS's three training modes—synchronous, asynchronous, and hybrid synchronous—accelerate distributed DNN training by approximately 8.2x, 19x, and 18.7x on 16 nodes, respectively, compared to a single node. In particular, the EDDIS hybrid synchronous training mode achieves training speeds that are 2.8 times faster than PyTorch DDP and 2.3 times faster than Horovod when training the Yolov5m model.**

***Index Terms*—Data-parallel Training, Deep Neural Network, Distributed Training, DNN, EDDIS, Heterogeneous GPU, Hybrid-EASGD, SoftMemoryBox, SSGD,**

I. Introduction

As deep-learning continues to advance, there is an increasing demand for models with more capabilities (multimodal) and higher performance (accuracy). Deep neural network models that generally offer a higher performance and more functionalities have more parameters [1]. Larger models must be trained with more data to achieve better performance. Consequently, training DNN models with superior capabilities and accuracy requires an exponential increase in computational power [2]–[4]. To address the rapidly increasing computational requirements, high-performance computing(HPC) systems must be used to train DNNs in a distributed manner. The distributed training of DNN models using HPC systems requires a very high communication bandwidth among the distributed processes. Insufficient bandwidth can lead to communication bottlenecks and prevent efficient distributed DNN training. Furthermore, as the number of distributed training processes (each process typically occupies one GPU) increases, the communication bottlenecks become more pronounced. Thus, the overall training time does not decrease linearly, and as the scale of distributed training exceeds a certain level, the training time increases. The rapid advancement and high cost of GPUs have led institutions to build fragmented, small-scale, heterogeneous GPU servers incrementally. Except for TensorFlow, the majority of open-source deep-learning libraries, such as Horovod and PyTorch DDP, support only synchronous training methods that perform better in homogeneous GPU environments. Therefore, they cannot effectively perform distributed training in environments that utilize heterogeneous GPUs owing to straggler problems.

In this study, we propose EDDIS, which reduces the scalability limitations and inefficiencies in distributed DNN training on heterogeneous GPU clusters, enabling faster and more cost-effective deep-learning model development. EDDIS is the successor version of ShmCaffe [5], which is a distributed deep learning technique that extends Caffe using SoftMemoryBox [6], [7] and TensorFlow based on SoftMemoryBox(TFSM) [8], a distributed deep-learning library based on TensorFlow v1.x. EDDIS supports data-parallel distributed training of DNN models written in TensorFlow v2.x. and PyTorch v1.x. It can significantly reduce the training time of computationally intensive DNNs with large amounts of training data by efficiently performing distributed training on heterogeneous GPU servers.

EDDIS is also more convenient than the conventional parameter server method for developing asynchronously distributed training codes. In general, asynchronously distributed training codes using the parameter server approach require separate implementation of code logic for the parameter servers and training workers with manual connection configurations between the parameter servers and workers in the source code, and they need to be executed separately individually. However, EDDIS enables all training workers to run simultaneously via the Message Passing Interface (MPI) library, whereas SoftMemoryBox (SMB) servers run as background daemons, allowing for easy initiation and termination

This work was supported by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean government[24ZS1100, Research on High-Performance Computing to overcome Limitations of AI]
* Corresponding author: 2sookwang@etri.re.kr

979-8-3503-6248-0/24/$31.00 ©2024 IEEE
Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

Darell Long, Ph.D.
09/17/2025
Exhibit I
H. Bautista, CSR #11600

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 1 of 10

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 368 of 411

of the training processes. Additionally, during distributed training, stopping a single mpirun process terminates all the training worker processes in the batch. Unlike the conventional parameter server model, the SMB servers in EDDIS operate as daemons and support distributed training for other DNNs.

The contributions of this study are as follows:

1) EDDIS can support data-parallel distributed training in various hardware environments, including homogeneous GPU and heterogeneous GPU environments, by providing three training modes that enable customized utilization of computing resources.
2) EDDIS offers high-speed asynchronous parameter sharing through global parameter sharding across multiple SoftMemoryBox servers and particularly achieves efficient GPU utilization capabilities by controlling the number of trained mini-batches per GPU differently based on the computing capacity.
3) EDDIS provides a fully parallelized technique between the threads responsible for performing training and updating global weights, which helps alleviate communication bottlenecks when training in a heterogeneous GPU environment using asynchronous/hybrid approaches. This technique allows the communication time to be masked as much as possible by computational time.
4) We propose Hybrid-EASGD (HEASGD), a new distributed training algorithm that improves asynchronous training performance over the Shared Memory-based EASGD (SEASGD) algorithm [5].

The remainder of this paper is organized as follows. Section II describes the background of the key concepts in this study. Section III introduces EDDIS. Section IV presents a comprehensive guide for training DNNs using EDDIS in a distributed manner. Section V presents the performance evaluation results of EDDIS. Finally, Section VI summarizes our study and discusses future research.

II. Background

Distributed deep learning is performed using data parallel, model parallel, or both. Data parallelism is an approach in which multiple workers train replicas of DNNs using a subset of the training dataset. Model parallelism is an approach in which multiple workers train a portion of the DNNs with the entire dataset [2], [9]–[12]. This paper discusses distributed deep learning with a focus on data parallelism, which is one of the two main methods.

In data-parallel training, distributed workers must periodically synchronize newly explored parameters with other workers, typically after each mini-batch training(i.e., step in PyTorch). Global parameter synchronization algorithms can be broadly classified into two categories: Synchronous Stochastic Gradient Descent (SSGD) and Asynchronous Stochastic Gradient Descent (ASGD). In the SSGD method, gradients are exchanged using all-reduce, and local weights are updated, which can lead to synchronization overhead [13], [14]. By contrast, the ASGD method uses a parameter server to receive gradients from workers, update global weights, and send the updated weights back to the workers. Although ASGD allows faster training with less synchronization overhead, it may take longer to achieve effective learning owing to gradient staleness [15]. EASGD is a model-averaging method in which each deep-learning worker trains a copy of the DNN model and exchanges weights with a parameter server to update the local weights [16], [17]. Stale Synchronous Parallelism (SSP) blends SSGD and ASGD by synchronizing gradients within a set delay, reducing the impact of outdated information [18], [19]. Chen's research showed that ASGD converges more slowly and yields lower accuracy than SSGD, making SSGD the preferred method for large-scale distributed deep learning [14]. Recent SSGD studies have improved accuracy and reduced communication overhead [20]. SSGD is most effective in homogeneous GPU clusters, particularly with Tesla GPUs that offer superior communication capabilities such as GPUDirect RDMA(Remote Direct Memory Access). However, in heterogeneous GPU clusters, the straggler problem can lower GPU utilization. HetSeq [21] approached this problem by using different batch sizes for heterogeneous GPUs and averaging losses based on the batch size to calculate gradients. However, HetSeq does not fully solve the straggler problem, and its need for tight integration with the model code significantly reduces its usability. Therefore, the training modes should be dynamically adjusted based on GPU resources: SSGD for homogeneous clusters, ASGD for heterogeneous clusters, and a combination of both, when different GPU types are used at the node level. ShmCaffe [5] combines the parameter server approach of DistBelief [10] with the shared memory access method of Hogwild [22]. SoftMemoryBox [6], [7] provides a remote shared memory buffer to store global weights among distributed workers, thereby creating the illusion of lock-free data access. However, unlike Hogwild, because the actual data reside in remote memory, updating weights without locks can lead to data races, potentially causing a loss of learned data. To address this issue, SoftMemoryBox ensures data consistency on the server side. Unlike traditional parameter servers, SoftMemoryBox does not depend on specific model network information and operates as a daemon that supports various models. Therefore, distributed workers must request updates directly from the global weights in SoftMemoryBox at a low level (e.g., tensor). EDDIS was developed based on this need and built upon SoftMemoryBox. The zero-redundancy optimizer (ZeRO) [23] is a technique that overcomes GPU memory limitations, requiring sufficient communication bandwidth between the GPU and the host memory or SSD. ZeRO is only effective for Tesla GPUs. Although open-source libraries such as DeepSpeed [24], Megatron-LM [25], and AxoNN [26] utilize ZeRO, this study does not include a direct comparison with ZeRO-based libraries because it targets heterogeneous GPUs such as GTX, RTX, and Tesla GPUs.

III. EDDIS

A. Software architecture

EDDIS is a high-speed data-parallel deep learning library based on SoftMemoryBox [7] designed for efficient distributed

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 2 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 369 of 411

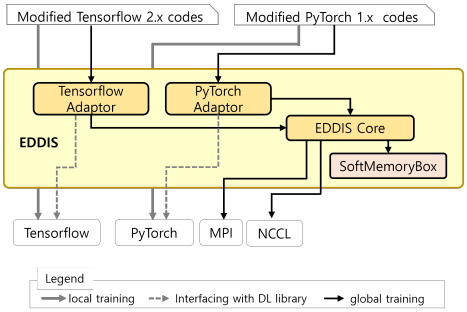


Fig. 1. Software architecture of EDDIS

DNN training in heterogeneous GPU environments. Fig. 1 shows the high-level software architecture of EDDIS. It utilizes SoftMemoryBox servers for parameter exchange and operates in conjunction with TensorFlow or PyTorch. To train DNNs in a distributed manner using EDDIS, some code must be modified using the EDDIS API. TensorFlow/PyTorch handles local parameter training, while EDDIS manages global parameter training. EDDIS comprises TensorFlow and PyTorch adapters, EDDIS Core module, and SoftMemoryBox. The adapters provide APIs to modify the original code for distributed training, whereas the EDDIS Core module handles parameter exchanges and updates, working with SoftMemoryBox EDDIS Core also adjusts the training load based on the processor's computational power. EDDIS uses the SoftMemoryBox framework [6], [7] to exchange and update global parameters, and employs multiple SMB servers and RDMA to reduce communication bottlenecks in InfiniBand networks.

EDDIS utilizes SMB for asynchronous and hybrid parameter updates. The SMB framework creates an environment similar to a virtual shared memory buffer by pooling the available memory segments from remote nodes for distributed processes. Fig. 2 illustrates the parameter update method using SoftMemoryBox for distributed training in EDDIS. EDDIS consolidates all DNN parameters into a single large tensor, with the SMB framework allocating a virtual shared buffer of matching size. By pooling equally sized memory segments from remote servers, this buffer enables seamless global parameter sharing across distributed training processes. EDDIS avoids communication bottlenecks and optimizes bandwidth utilization by evenly distributing memory segments across a scalable set of SMB servers, thereby dynamically increasing the number of servers required to satisfy the bandwidth requirements for training.

B. Data-parallel training modes

EDDIS supports three data-parallel training modes for distributed training. The manner in which the distributed processes update their local and global weights varies, depending on the training mode. First, EDDIS supports a synchronous training mode using the Synchronous Stochastic Gradient Descent (SSGD) algorithm such as Horovod, PyTorch Distributed Data Parallel(DDP), or TensorFlow distribution. Fig. 3 a) illustrates the operation of the synchronous training

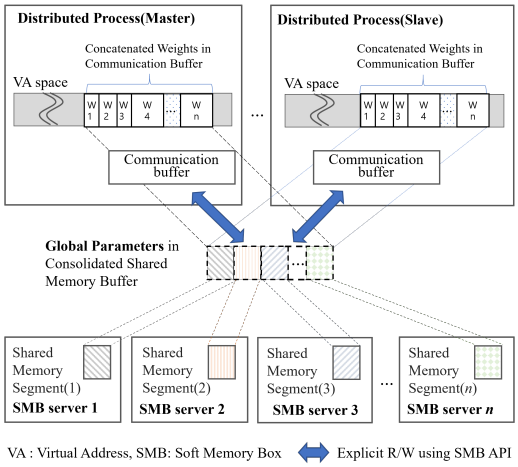


Fig. 2. The consolidated shared memory buffers are used for communicating global parameters among distributed processes.

mode(SYNC). Each distributed deep-learning process, which occupies one GPU in a computing node, trains a mini-batch with all the parameters. After calculating the gradients from the minibatch, the processes average the gradients using the AllReduce communication API of NVIDIA Collective Communications Library(NCCL) or MPI. The local weights are then updated using the averaged gradients. All processes start with the same initial weights and update the weights with the same averaged gradients; therefore, the weights of all the processes have the same values. Second, EDDIS supports an asynchronous training mode(ASYNC), as shown in Fig. 3). The asynchronous mode is particularly suitable for distributed training when GPUs are heterogeneous. It is also an algorithm that can be used in homogeneous GPU environments. Finally, EDDIS supports the hybrid training mode(HSYNC), as shown in Fig. 3 c). The HSYNC model combines the benefits of both SYNC and ASYNC training methods. In this mode, each distributed-training worker belongs to a specific subgroup. For example, in Fig. 3 c), the distributed processes within a single computing node belong to the same subgroup. The key feature of this approach is that training is performed synchronously within each subgroup, whereas training between subgroups is performed asynchronously. Fig. 3 c) illustrates the operation of the hybrid algorithm. After completing mini-batch training, the distributed processes within each subgroup exchange and average gradients using NCCL AllReduce, and then update their local weights based on the averaged gradients. The representative processes of each subgroup read the global weights from the SoftMemoryBox server, update the local weights, and then update the global weights using the differences between them.

C. Distributed optimizer algorithms

EDDIS supports a few parameter-update algorithms, such as SSGD, ASGD, SEASGD, and HEASGD. The SSGD algorithm is the only parameter update algorithm for the SYNC training mode. The others are used in the ASYNC and HSYNC

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 3 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 370 of 411

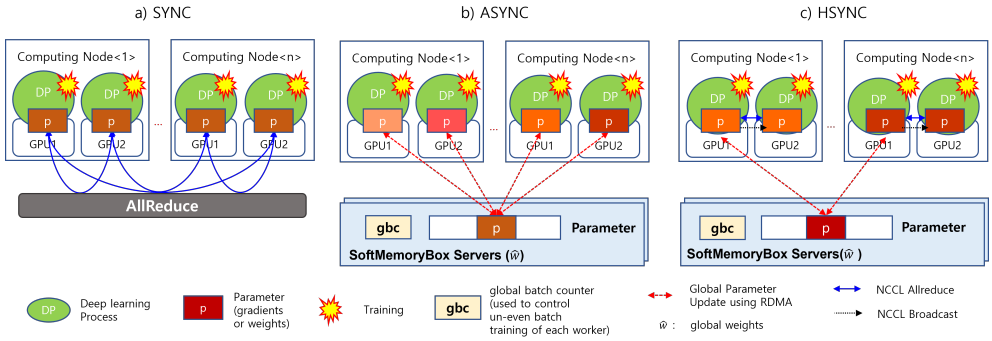


Fig. 3. Comparison of the three data-parallel training modes of EDDIS

training modes. Equation (1) is the local and global weight update formula for SSGD (w': updated weights, w: weights, η is the learning rate, $\hat{g}$ is the averaged gradient).

$$w' = w - \eta\hat{g} \tag{1}$$

Equation (2) describes the manner in which the global ($\hat{w}$) and local weights (w_x) are updated in the ASYNC training mode(g_x: gradients of x workers).

$$\begin{aligned} \hat{w}' &= \hat{w} - \eta g_x \\ w_x' &= \hat{w}' \end{aligned} \tag{2}$$

Equation (3) expresses the SEASGD algorithm [5]. Conceptually, this algorithm updates local and global weights using the EASGD algorithm. After computing the local gradients from each minibatch training, each distributed process updates its local weights($w_x - \eta g_x$) using a locally computed gradient(g_x). Second, it updates the local weights using the weight difference ($\Delta w = w_x - \hat{w}$) between the global weights ($\hat{w}$) and local weights(w_x). After the local weights are updated, the weight difference(Δw) is uploaded to the SoftMemoryBox servers and used to update the global weights ($\hat{w}'$). The moving rate(α) is a hyperparameter that represents the extent to which the local and global weights are updated immediately from the weight differences(Δw). The SEASGD algorithm generally uses a tuned value for the moving rate(α) starting from the reciprocal of the number of workers (N), which is $1/N$.

$$\begin{aligned} w_x' &= w_x - \eta g_x - \alpha\left(w_x - \hat{w}\right) \\ \hat{w}' &= \hat{w} + \alpha\left(w_x - \hat{w}\right) \end{aligned} \tag{3}$$

Equation (4) illustrates the newly proposed HEASGD algorithm, which is a variant of SEASGD. The local weight update equation for HEASGD is the same as SEASGD. However, the value of the moving rate(α) is applied different from that used in SEASGD. The moving rate(α) is applied with the value of $(N-1)/N$. Conceptually, the global weight update of HEASGD in Equation (4) combines the global weight update methods of ASGD and SSGD. The difference from the global weight update method of ASGD in Equation (2) is that the gradients are divided by the number of workers(N) before updating the global weights.

$$\begin{aligned} w_x' &= w_x - \eta g_x - \alpha\left(w_x - \hat{w}\right) \\ \hat{w}' &= \hat{w} - \frac{\eta g_x}{N} \end{aligned} \tag{4}$$

Equation (5) represents the formula for sequential global weight updates by N workers along the time axis, starting from time *t*.

$$\begin{aligned} \hat{w}^{t+1} &= \hat{w}^t - \frac{\eta g_x^t}{N} \\ &\ldots \\ \hat{w}^{t+N} &= \hat{w}^{t+N-1} - \frac{\eta g_x^{t+N-1}}{N} \end{aligned} \tag{5}$$

Equation (6) is the integrated formula derived from Equation (5). When the formulas for the global weight updates by N workers are combined in Equation (6), it is similar to the SSGD algorithm in Equation (1), which updates the global weights using the average weights of N workers. However, in the SSGD algorithm, each worker averaged the gradients learned from the same weights, whereas in the HEASGD algorithm, each worker averaged the gradients learned from slightly different weights. We estimated that the averaged gradients($\hat{g}$) of SSGD and the averaged gradients($\tilde{g}$) of HEASGD are very similar($\tilde{g} \approx \hat{g}$). Through extensive experimentation, we aim to prove that the aforementioned estimation enables the scaling of asynchronous distributed training.

$$\hat{w}^{t+N} = \hat{w}^t - \frac{\eta(g_x^t + g_x^{t+1} + \ldots + g_x^{t+N-1})}{N} = \hat{w}^t - \eta\tilde{g} \tag{6}$$

D. Full-parallel and uneven-batches training

EDDIS supports SoftMemoryBox-based, full-parallel, and uneven-batch trainable asynchronous parameter updates. EDDIS improves upon ShmCaffe [5], where computation and communication are performed in a half-parallel manner by enabling full-parallel global weight updates. In addition, it introduces the capability of distributed workers running on heterogeneous GPUs to train an uneven number of batches during each epoch. In Fig. 3 b) and c), it can be observed that a global batch counter(gbc) is allocated in the SoftMemoryBox servers and shared among the workers. This counter is atomically shared among workers via InfiniBand RMDA atomic operations, and is used to control uneven batch training.

Fig. 4 illustrates the procedure between the main thread responsible for local training and the GWU thread that handles global weight updates in SEASGD and

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 4 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 371 of 411

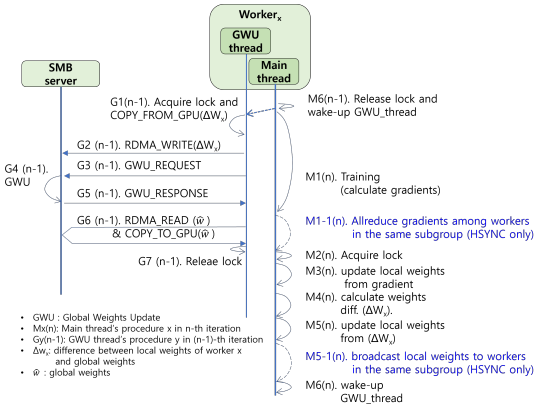


Fig. 4. The procedure of the full-parallel training and parameters updating for SEASGD and HEASGD algorithms

HEASGD algorithms. Specifically, during step G1(n-1), the COPY_FROM_GPU(Δw_x) phase involves reading Δw_x from the GPU memory of worker x into host memory. In step G6(n-1), the most recent global weights are read in advance and copied to the GPU's global weights buffer. Computation and communication were parallelized by having the GWU thread performing RDMA_READ($\hat{w}$) & COPY_TO_GPU($\hat{w}$), the computation and communication are fully parallelized. In general, the training time of the M1(n) phase is longer than that of the entire G1-7 phases. However, in cases where the time of G1-7 phases is longer than the training time of M1(n) owing to large parameters or limited network bandwidth, the main thread may need to wait until the completion of the G1-7 phases in the GWU thread. Both threads employ control locks to control the flow between them.

The procedure in Fig. 4 was applied in HSYNC training mode. However, gradient averaging among the same subgroups of workers was included, such as M1-1(n). The calculation of weight differences (Δw_x), second update of local weights, and communication with the SoftMemoryBox server for updating global weights($\hat{w}$) were performed by a representative worker of the subgroup. The representative worker broadcasts the secondarily updated local weights to the other distributed workers within the same subgroup using NCCL/MPI Broadcast in phase M5-1(n), enabling them to proceed with the training of the next mini-batch. The ideal hardware environment for applying this hybrid algorithm is when the GPU servers internally have the same type of GPUs but there are different types of GPUs across servers.

E. EddisOptimizer

The Optimizer class in TensorFlow or PyTorch is crucial for computing gradients and updating weights. EddisOptimizer is a wrapper function that takes an instance of the original optimizer class, creates a new class that extends it based on the training mode, and returns an instance of this new class. As shown in Fig. 5, EddisOptimizer outperforms the methods based on the training mode.

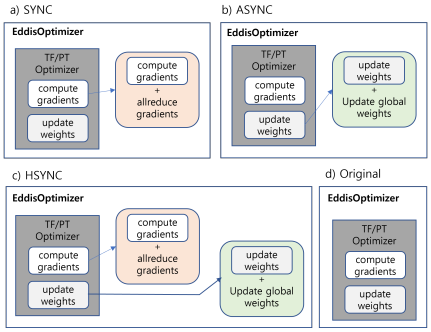


Fig. 5. The EddisOptimizer based on training modes and environment.

- SYNC mode (Fig. 5 a)): Overrides *compute gradients* method to perform an all-reduce operation that averages gradients among all distributed processes.
- ASYNC mode (Fig. 5 b)): Extends *update weights* method to perform a second update based on the difference between local and global weights, and triggers an update of the global weights.
- HSYNC mode (Fig. 5 c)): Overrides both *compute gradients* and *update weights* methods, with the all-reduce operation averaging gradients only among distributed processes within the same subgroup.

If training is not distributed, EddisOptimizer returns the original optimizer instance without modifications (Fig. 5 d)).

IV. Distributed Training using EDDIS

To perform data-parallel distributed training using EDDIS, researchers must modify the code for training a DNN model according to the manual modification procedures of EDDIS. The basic modification procedures for TensorFlow and PyTorch training codes are as follows.

A. Initializing and preparing distributed training

During the initialization of EDDIS, different parameter communication threads are launched depending on the training mode, with SMB initialization handled accordingly. In the SYNC mode, each process runs a main thread for local parameter training and a gradient communication thread that exchanges gradients during backpropagation to overlap communication with computation. SMB is not initialized in SYNC mode. The SMB should be initialized in the ASYNC or HSYNC modes. Each process starts with a global weight update (GWU) thread (Fig. 4) to establish reliable InfiniBand connections to the SMB servers using the SMB API [7]. Additionally, a global batch counter (gbc) is allocated and initialized to zero.

1) Initializing codes: First, the *eddis.tensorflow* or *eddis.torch* packages initialize the EDDIS framework by calling *eddis.init()*. Users can specify the data-parallel training mode and moving rate for ASYNC/HSYNC modes as arguments of *eddis.init()*.

```
eddis.init(training_mode=eddis.HSYNC, sgd_type=eddis.HEASGD
    , moving_rate=0.75).
```


Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 5 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 372 of 411

2) Adjusting learning rate: The initial learning rate in the existing training code was optimized for training on a single GPU. Generally, in the SSGD training mode, as the number of distributed training processes increases from *1* to *n*, the batch size increases proportionally. Consequently, the learning rate should also be scaled by the same factors. If the training mode is ASYNC, it is not necessary to adjust the learning rate. If the training mode is HSYNC, rather than multiplying the learning rate by the total number of distributed processes *n*, users scale it by the number of processes within each subgroup, denoted as *m*. For instance, a subgroup may consist of all GPUs within a single node.

3) Wrapping Optimizer: Developers or researchers need to wrap the existing TensorFlow or PyTorch optimizer into the EddisOptimizer, allowing it to support the three training modes in EDDIS while preserving the default behavior of the original optimizer. Thus, the functions for the *compute gradients* and *update weights* of the original optimizer are selectively expanded based on the specified training mode in the initialization phase.

```
optimizer = optimizers.SGD(learning_rate=0.01*n, momentum
    =0.9) # adjusted learning rate
optimizer = eddis.EddisOptimizer(optimizer) #Wrap optimizer
```

B. Controlling training loop

TensorFlow uses *tf.data.Dataset* or *tf.data.Iterator*, whereas PyTorch uses *torch.utils.data.DataLoader* to feed training data. If these classes are not used as repeat options, they stop providing mini-batches after the total number of steps required for an epoch. This study proposes a training progress control technique to enhance training efficiency in heterogeneous GPU environments. In the SSGD mode, when scaling to *n* training processes, each process trains the same number of mini-batches ($total\_steps\_per\_epoch/n$). However, this method is inefficient when GPUs have different computational powers. To address this, we propose a technique that adjusts the training steps such that faster GPUs train more mini-batches, whereas slower GPUs train fewer mini-batches. This allows each distributed process to train a different number of mini-batches, thereby efficiently utilizing heterogeneous GPUs. For training codes that utilize a Data Iterator or *DataLoader*, the *DataGenerator* class is used to provide a variable number of minibatches during an epoch instead of a fixed number. This allows the distributed processes to train different numbers of minibatches. The EDDIS *DataGenerator* class is instantiated with arguments, such as the optimizer, data source, and steps_per_epoch, enabling it to efficiently supply varying mini-batches to the distributed processes. The *DataGenerator* instance shares the total number of minibatches trained by all distributed processes through a global batch counter on the SoftMemoryBox server. It calculates the number of averaged mini-batches trained, and continues feeding mini-batches until the average count reaches the steps_per_epoch specified as an argument.

1) Instantiating DataGenerator: To effectively utilize heterogeneous GPUs in ASYNC or HSYNC modes, users can optionally create *DataGenerator* instances as follows:

```
data_generator = eddis.DataGenerator(optimizer, data_source
     = train_data_iterator, steps_per_epoch =
    iterations_per_epoch // eddis.size())
```

2) Modifying training loop: To enable heterogeneous GPU-based distributed training, as previously mentioned, users must modify the training loop using an instance of DataGenerator.

```
for i (images and labels) in data_generator:
```

C. Other modification procedures

1) Modifying validation codes: Typically, training codes include codes for evaluating the trained model on the validation data at specific intervals, such as after each training epoch. Because it is not necessary for all distributed processes to execute these validation codes, users can restrict them to only the master process. In this case, although the training time is reduced to 1/n using distributed training, the validation time remains unchanged. However, it is possible to reduce the training time further by dividing the validation computation to distributed processes.

2) Restricting codes for master process: The training activities that should be performed only by the master (or root) process include saving or loading the checkpoint, standard output and logging, and validation.

D. Starting and stopping distributed training

EDDIS runs distributed DNN training processes concurrently using MPI, unlike PyTorch DDP and TensorFlow, where each distributed process must be launched separately. The *mpirun* command in MPI allows running a large number of distributed processes simultaneously, with the added advantage of terminating them all at once if necessary during distributed training, such as when DNN training is not progressing as expected because of incorrect hyperparameters. Furthermore, in the event of abnormal termination of any distributed process, the remaining processes can be terminated simultaneously, preventing a scenario in which the DNN training stalls while still occupying GPU resources.

V. Experimental Study

A. Environments

1) Hardware environments: We used nine Supermicro 4028GR-TRT2 servers, four 4124GS-TNR servers, two 7048GR-TR servers, and an ASUS ESC4000A server. The 4028GR-TRT2 servers are equipped with two Intel Xeon E5-2690 v4 CPUs (14 cores, 2.3 GHz), 128 GB DDR4-2400 MHz memory, and four Nvidia GPUs. The 4124GS-TNR servers featured two AMD EPYC™ 7343 CPUs (16 cores, 3.2 GHz) and 256 GB DDR4-3200 MHz memory. The 7048GR-TR servers had two Intel Xeon E5-2620 v4 CPUs (8 cores, 2.1 GHz) and 128 GB DDR4-2400 MHz memory. The ESC4000A server has an AMD EPYC 7313P CPU (16 cores, 3.0 GHz) and 128 GB DDR4-3200 MHz memory. The GPUs used were

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 6 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 373 of 411

NVIDIA GTX Titan Xp, GTX Titan V, RTX 2080TI, RTX Titan, RTX 3090, Tesla P100, and Tesla A100. Each server was connected via an HDR InfiniBand HCA (200 Gbps) to a Mellanox QM8700 InfiniBand switch.

2) Software environments: The operating system was Ubuntu 20.04, and the Linux kernel version was 5.4.0-135-generic. We used CUDA 11.4 (including cudnn 8.2.4), TensorFlow 2.5, PyTorch 1.10.1, EDDIS v1.5 and USMB v2.0.

3) DNN & dataset used in the experiments: For this experiment, we selected two convolutional neural networks (CNNs): ResNet50 [27] for image classification and Yolov5 [28]–[31] for object detection. ResNet50 is one of the most widely used DNNs for evaluating the parallel training of distributed data [32]–[35]. Many researchers have demonstrated their work by showcasing the results of the distributed training of ResNet50 using the ILSVRC 2012 ImageNet dataset [36]. In this study, YOLO, which has gained the widest user base as an object detection model, was evaluated by conducting distributed training on the YOLOv5 medium [31] using the COCO dataset [37].

B. Comparison of training time among three training modes

Fig. 6 compare the training times of ResNet50 implemented in TensorFlow using the three EDDIS training modes: SYNC, ASYNC, and HSYNC. The experiment began with a single node and was scaled up to 16 nodes, doubling the number of nodes at each step. The graph shows the training times for 100 epochs for the Imagenet1K dataset.

Fig. 7 compares the training times for YOLOv5m implemented in PyTorch for the three EDDIS training modes. This graph shows the training times for 300 epochs on the COCO dataset, using the same hardware environment as that in Fig. 6. Uniform batch sizes were used in all the experiments (ResNet50: 64, YOLOv5m: 16), and SGD was used as the local optimizer. There were five for ResNet50 and three for YOLOv5m, with a weight decay of 5e-4 for YOLOv5m. The learning rates, weight decay (for ResNet50), and moving rates (only for ASYNC and HSYNC) were tailored to the node count for each experiment, as listed in Tables I and II.

TABLE I
HYPER-PARAMETERS FOR RESNET50 TRAINING

Training mode	weight decay	learning rate(moving rate) (#Nodes)				
		1	2	4	8	16
SYNC	1e-5	0.4	0.8	1.6	3.2	6.4
ASYNC	1e-4	0.1 (0.75)	0.2 (0.875)	0.2 (0.938)	0.2 (0.969)	0.2 (0.984)
HSYNC	1e-4 1e-4	0.2 (-)	0.2 (0.5)	0.2 (0.75)	0.4 (0.875)	0.6 (0.938)

SYNC with SSGD and ASYNC/HSYNC with HEASGD

TABLE II
HYPER-PARAMETERS FOR YOLOV5M TRAINING

Yolov5m	learning rate	Moving rate (#Nodes)				
		1	2	4	8	16
SYNC	0.01x#N[1]	-	-	-	-	-
ASYNC	0.01	0.25	0.125	0.0625	0.0625	0.035
HSYNC	0.01x4		0.5	0.25	0.15	0.1

[1] #N : Number of Nodes(each node has 4 GPUs)

The experiment was conducted using 16 nodes with heterogeneous GPUs. The baseline training time for a single node was determined by averaging the times across all the nodes, whereas for two, four, and eight nodes, the average training time was calculated by grouping nodes with different heterogeneous GPU combinations into two or more sets. Fig. 6 and Fig. 7 displays the training speedup as a line graph, illustrating the improvement when using 2, 4, 8, and 16 nodes compared to a single node.

1) Synchronous training: According to Fig. 6 a), and Fig. 7 a), the training time for ResNet50 using the SYNC mode on one node was approximately 59 h and that for YOLOv5m was 62 h and 53 min. With 16 nodes, the training times for ResNet50 was reduced to approximately 7 h and 1 min and for YOLOv5m, it was 7 hours and 48 min, respectively. times higher than speedups of 8.39 times for ResNet50 and 8.06 times for YOLOv5m. Despite utilizing 16 times the resources, the scaling efficiencies were approximately 52.4% and 50.3%, respectively, owing to GPU heterogeneity.

2) Asynchronous training: Fig. 6 b) and Fig. 7 b) shows the results obtained using the ASYNC training mode. For 16 nodes, the training times for ResNet50 was approximately 3 h and 13 min, whereas for YOLOv5m, it was 3 h and 6 min, respectively. This corresponds to speedups of 20.1 times for ResNet50 and 17.8 times for YOLOv5m, with scaling efficiencies of 125.5% and 111%, respectively. Scaling efficiency cannot technically exceed 100%, but in this experiment, it did due to the "fallacy of average". The baseline was set using the average training speed across all the nodes, which underestimated the potential performance. During scaling, faster GPUs handled more work, resulting in scaling efficiencies that exceed 100%. This outcome was further attributed to EDDIS's ability of EDDIS to effectively utilize heterogeneous GPU nodes through uneven batch training, leading to even higher scaling efficiencies.

3) Hybrid training: Fig. 6 c) and Fig. 7 c) shows the results for the HSYNC mode. On 16 nodes, the training times for ResNet50 was approximately 3 h and 22 min and for YOLOv5m, it was 3 h and 9 min, respectively. This results in speedups of 17.5 times for ResNet50 and 19.9 times for YOLOv5m compared to a single node. The scaling efficiencies were approximately 109.3% and 124.5%, respectively. HSYNC mode is more than twice as fast as SYNC mode, with training times on 16 nodes being 2.08 times and 2.46 times faster for ResNet50 and YOLOv5m, respectively.

C. Comparison of scaling performance between SEASGD and HEASGD algorithms

Through our experiments, we confirmed that the deep-learning performance (e.g., accuracy) of the previously proposed SEASGD algorithm degraded significantly as the number of workers increased. Fig. 8 a) depicts the training performance (Top-1 Accuracy) of ResNet50 when trained using the SEASGD algorithm in ASYNC training mode. As the number of training workers increased from one node(4GPUs) to 16 nodes(64GPUs), the training performance degraded significantly. A previous study mitigated this issue to a certain extent by introducing the HSYNC training mode [5]. This

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 7 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 374 of 411

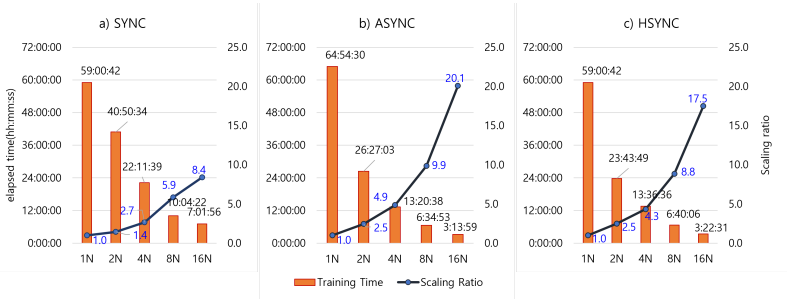


Fig. 6. Comparison of EDDIS (TensorFlow) Training Time of ResNet50

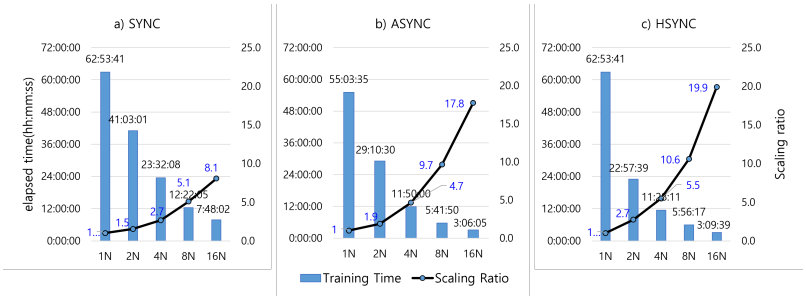


Fig. 7. Comparison of EDDIS (PyTorch) Training Time of Yolov5

is because in HSYNC mode, the training workers are sub-grouped, reducing the number of workers updating the global weights asynchronously from 64 to 16. Fig. 8 b) illustrates the training performance when using the SEASGD algorithm in HSYNC mode. It can be observed that in a 16-node, 64-GPU environment, the performance degradation in ASYNC mode was 14.1%, whereas in HSYNC mode, it decreased by only 0.68% as shown in Fig. 8 b). However, even when training in HSYNC mode, there is an increased performance degradation as the scaling factor increases. This study addresses this issue by using the newly proposed HEASGD algorithm. Fig. 8 c) shows the performance when trained using the HEASGD algorithm in ASYNC training mode. In a 16 nodes 64 GPUs environment, the HEASGD algorithm experienced a 1.08% degradation in training performance. As seen in Fig. 8 d), in HSYNC mode, the performance degradation is limited to only 0.27%. Fig. 9 directly compare the SEASGD and HEASGD algorithms. Looking at Fig. 9 a), it can be seen that the HEASGD algorithm is 12.3% better than the SEASGD algorithm in ASYNC training mode, and Fig. 9 b) shows that HEASGD outperformed SEASGD by 0.41% in HSYNC training mode.

D. Comparison of accuracy among three training modes

Fig. 10 compares the top-1 accuracy of ResNet50 across the three training modes: SYNC, ASYNC, and HSYNC. In the SYNC mode, the SSGD algorithm was used, whereas HEASGD was employed in both ASYNC and HSYNC modes. As the number of workers increased, the training performance tended to degrade across all training modes. According to Fig. 10, in SYNC mode, the training performance decreases by 0.34% as the number of nodes increases from 1 to 16. Although ASYNC mode shows a better performance than the

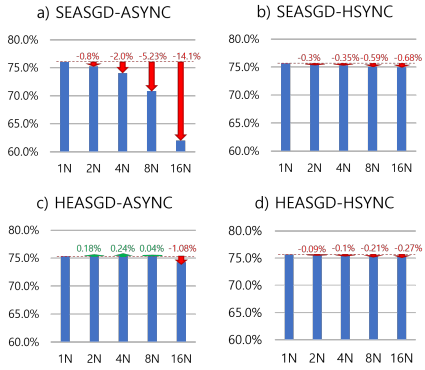


Fig. 8. Comparison of scaling performance of distributed optimizer algorithms(SEASGD and HEASGD) in ASYNC and HSYNC training modes

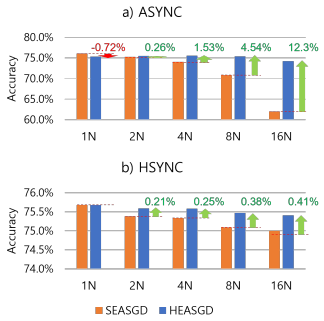


Fig. 9. Direct Comparisons of performance between SEASGD and HEASGD algorithms

SYNC mode with two, four, and eight nodes, its performance drops below that of the SYNC mode when the number of nodes it reaches is 16. By contrast, HSYNC mode consistently

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 8 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 375 of 411

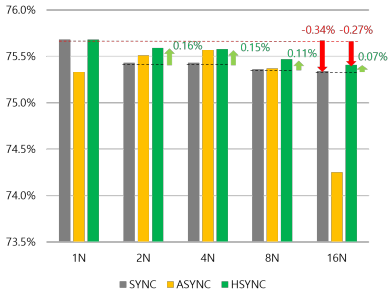


Fig. 10. Comparison of EDDIS Validation Results(Top-1 Accuracy)

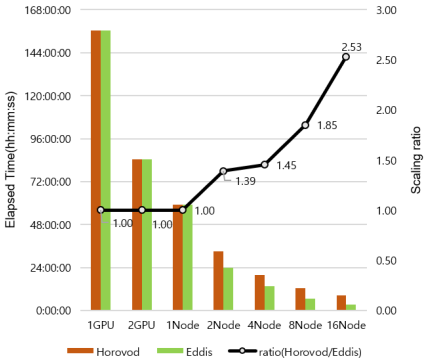


Fig. 11. Comparison of Tensorflow Resnet50 Training Time between Horovod and EDDIS(HSYNC)

achieves better performance than the SYNC mode, even as the number of nodes increases. This experiment was conducted to compare the performances across the three training modes. The lower performance observed compared with the model's best-known performance is due to the lack of optimizations, such as data augmentation, in the training process.

E. Training comparison of EDDIS, Horovod, and DDP

Fig. 11 compares the training times of the Tensorflow ResNet50 model with those of the baseline(1GPU), Horovod (HVD), and EDDIS(HSYNC) models when training the ResNet50 model on the ImageNet1K dataset for 100 epochs. EDDIS completed training 46.36 times faster than the baseline, with a scalability efficiency of 72.43%. EDDIS also achieved 2.53 times faster as Horovod in a 16-node, 64-GPU environment. The final training accuracy was 76.63% for the baseline, 75.42% for EDDIS, and 73.4% for Horovod, with EDDIS showing a 2% better training performance than Horovod.

Fig. 12 a) compares the training times of the YOLOv5m model among the baseline(1GPU), PyTorch Distributed DataParallel(DDP), Horovod(HVD), and EDDIS(only SYNC mode). Fig. 12 b) compares the training times of the YOLOv5m model using the three training modes, DDP, HVD and EDDIS, on 16 nodes with 64 heterogeneous GPUs. The EDDIS SSGD training mode achieved a similar training time to Horovod and was 1 h and 20 min faster than DDP. EDDIS HSYNC training mode is 2.8x and 2.3x faster than DDP and HVD respectively in training the Yolov5m model with 300 epochs on the COCO dataset. We achieved a similar training

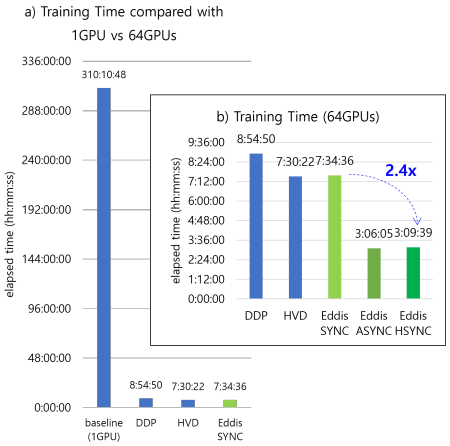


Fig. 12. Comparison of Yolov5m Training Time among a) PyTorch(1GPU, baseline), EDDIS(SYNC), DDP and Horovod(16 Nodes, 64 Heterogeneous GPUs), b) DDP, Horovod, EDDIS(SYNC, ASYNC, HSYNC)

performance (mAP) using EDDIS. When training the Yolov5m model, the baseline trained on one GPU achieved 0.441, DDP trained on a 16-node 64-GPU environment achieved 0.429, Horovod achieved 0.426, and EDDIS achieved 0.431.

VI. Conclusion

In this study, we propose EDDIS, which addresses the communication bottleneck in distributed deep neural network (DNN) training and efficiently utilizes heterogeneous GPUs to significantly speed up training. EDDIS offers three training modes—SYNC, ASYNC, and HSYNC—allowing the selection of the optimal mode based on the combination of available GPUs. This approach facilitates the distributed training of DNNs written in TensorFlow and PyTorch with minimal modifications to the existing code. EDDIS has strong node scalability and operates by increasing the number of SMB memory servers to secure the required bandwidth as the communication volume increases with the expansion of the training nodes. EDDIS also fully overlaps computation and communication times, significantly reducing the synchronization time between nodes.

EDDIS achieves approximately 8.2 times, 19, and 18.7 times faster training compared to using a single node when employing 16 nodes in the SYNC, ASYNC, and HSYNC modes, respectively. Notably, EDDIS(HSYNC) delivers training speeds that are 2.8x and 2.3x faster than DDP and Horovod, respectively. To address the scaling limitations of the existing SEASGD algorithm, we introduced a new HEASGD algorithm. The HSYNC mode in HEASGD further improves performance over the SYNC mode. EDDIS leverages MPI to enable concurrent deep learning process execution and termination, but requires uniform hyperparameters across processes. This limitation restricts the batch size to the GPU with the smallest memory in heterogeneous setups, potentially underutilizing GPUs with a larger memory. EDDIS partially addresses this by enabling the asynchronous training of more batches on GPUs with larger memory.

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 9 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 376 of 411

In future research, we plan to adapt asynchronous algorithms for heterogeneous GPU environments with varying numbers of GPUs per node and evaluate the performance of distributed training. Additionally, to address the convergence issues in the HSYNC method, we will enhance EDDIS scalability by extending the unit of subgroups from a single node to a multi-node configuration. Furthermore, experiments will be conducted on large language models (LLMs) such as BERT within the EDDIS framework, followed by further optimization based on the results.

REFERENCES

[1] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *CoRR*, vol. abs/2001.08361, 2020.

[2] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: building an efficient and scalable deep learning training system," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, (Broomfield, CO, USA), pp. 571–582, 2014.

[3] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on International Conference on Machine Learning (ICML'11)*, (Bellevue, Washington, USA), pp. 265–272, 2011.

[4] S. Bianco, R. Cadène, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64270–64277, 10 2018.

[5] S. Ahn, J. Kim, E. Lim, W. Choi, A. Mohaisen, and S. Kang, "Shmcaffe: A distributed deep learning platform with shared memory buffer for hpc architecture," in *Proceedings of the 38th IEEE International Conference on Distributed Computing System (ICDCS'18)*, (Vienna, Austria), July 2 - 5, 2018.

[6] S. Ahn, J. Kim, E. Lim, and S. Kang, "Soft memory box: A virtual shared memory framework for fast deep neural network training in distributed high performance computing," *IEEE Access*, vol. 6, pp. 26493–26504, 2018.

[7] S. Ahn and E. Lim, "Softmemorybox ii: A scalable, shared memory buffer framework for accelerating distributed training of large-scale deep neural networks," *IEEE Access*, vol. 8, pp. 207097–207111, 2020.

[8] E.-J. Lim, S.-Y. Ahn, Y.-M. Park, and W. Choi, "Distributed deep learning framework based on shared memory for fast deep neural network training," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1239–1242, 2018.

[9] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, (New York, NY, USA), p. 873–880, Association for Computing Machinery, 2009.

[10] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. W. Senior, P. A. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proceedings of 26th Annual Conference on Neural Information Processing (NIPS'12)*, (Lake Tahoe, NV, USA), pp. 1232–1240, December 3-6, 2012.

[11] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *CoRR*, 2014.

[12] W. Wang, G. Chen, H. Chen, T. T. A. Dinh, J. Gao, B. C. Ooi, K. Tan, S. Wang, and M. Zhang, "Deep learning at scale and at ease," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, vol. 12, no. 4s, pp. 69:1–69:25, 2016.

[13] J. Wang and L. Cheng, "Distdl: A distributed deep learning service schema with GPU accelerating," in *Proceedings of the 17th Asia-Pacific Web Conference on Web Technologies and Applications (APWeb'15)*, (Guangzhou, China), pp. 793–804, September 18-20, 2015.

[14] J. Chen, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous SGD," in *Proceedings of the International Conference on Learning Representations (ICLR) Workshop Track*, 2016.

[15] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B. Su, "Scaling distributed machine learning with the parameter server," in *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14)*, (Broomfield, CO, USA), pp. 583–598, October 6-8, 2014.

[16] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM J. Control Optim.*, vol. 30, p. 838–855, July 1992.

[17] S. Zhang, A. E. Choromanska, and Y. LeCun, "Deep learning with elastic averaging sgd," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 685–693, Curran Associates, Inc., 2015.

[18] S. Gupta, W. Zhang, and F. Wang, "Model accuracy and runtime tradeoff in distributed deep learning: A systematic study," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 171–180, 2016.

[19] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Comput. Surv.*, vol. 52, Aug. 2019.

[20] H. Mikami, H. Suganuma, P. U.-Chupala, Y. Tanaka, and Y. Kageyama, "Imagenet/resnet-50 training in 224 seconds," *CoRR*, vol. abs/1811.05233, 2018.

[21] Y. Ding, N. Botzer, and T. Weninger, "Hetseq: Distributed gpu training on heterogeneous infrastructure," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 15432–15438, May 2021.

[22] B. Recht, C. Ré, S. J. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Proceedings of the 25th Advances in Neural Information Processing Systems (NIPS'11)*, (Granada, Spain), pp. 693–701, 12-14 December 2011.

[23] J. Ren, S. Rajbhandari, R. Y. Aminabadi, O. Ruwase, S. Yang, M. Zhang, D. Li, and Y. He, "Zero-offload: Democratizing billion-scale model training," *CoRR*, vol. abs/2101.06840, 2021.

[24] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[25] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on gpu clusters using megatron-lm," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[26] S. Singh and A. Bhatele, "Axonn: An asynchronous, message-driven parallel framework for extreme-scale deep learning," 2023.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR'16)*, (Las Vegas, NV, USA), pp. 770–778, June 27-30, 2016.

[28] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[29] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018.

[30] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," 2022.

[31] J. Glenn, "Yolov5 release v6.1.." https://github.com/ultralytics/yolov5/releases/tag/v6.1.

[32] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch sgd: Training imagenet in 1 hour," *arXiv preprint arXiv:1706.02677*, 2017.

[33] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer, "Imagenet training in minutes," in *Proceedings of the 47th International Conference on Parallel Processing*, ICPP 2018, (New York, NY, USA), pp. 1:1–1:10, ACM, 2018.

[34] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, *et al.*, "Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes," in *Proceedings of Workshop on Machine Learning Systems in The 32th Annual Conference on Neural Information Processing Systems. IEEE*, 2018.

[35] C. Ying, S. Kumar, D. Chen, T. Wang, and Y. Cheng, "Image classification at supercomputer scale," *ArXiv*, vol. abs/1811.06992, 2018.

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition (CVPR) 2009. IEEE Conference on. IEEE*, 2009.

[37] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

Authorized licensed use limited to: Michael Heim. Downloaded on April 14,2025 at 18:36:17 UTC from IEEE Xplore. Restrictions apply.

ParTec EX2014
Microsoft v ParTec, IPR2025-00318
Page 10 of 10
ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 377 of 411

Disk Failure Prediction in Heterogeneous Environments

Darell Long, Ph.D.
09/17/2025
Exhibit J
H. Bautista, CSR #11600

Carlos A. Rincón C.*†, Jehan-François Pâris*, Ricardo Vilalta*, Albert M. K. Cheng* and Darrell D. E. Long‡
*Department of Computer Science, University of Houston, TX, Houston, USA.
†Department of Computer Science, Universidad del Zulia, Maracaibo, Venezuela.
‡Department of Computer Science, University of California, Santa Cruz, CA, USA.

***Abstract*—Recent studies have shown the benefits of using SMART attributes to predict disk failures in homogeneous populations of disks from the same make and model. We address here the case of data centers with more heterogeneous disk populations, such as the ones described in the BackBlaze datasets, and propose to build global disk failure predictors that would apply to disks of all makes and models. Our first challenge was the large number of SMART parameters that were missing for most makes and models in many disk instances of our dataset. As a result, we had to discard the SMART attributes that were missing in at least 90 percent of the disks, which left us with 21 SMART attributes. We then applied a Reverse Arrangement Test to these attributes to select the strongest disk failure indicators.**

We investigated three different machine learning models (Decision Trees, Neural Networks, and Logistic Regression) using the 2015 BackBlaze data to train and validate our predictors. Our best model was a decision tree that identified true failure events among the disks that tested positive for at least one of our failure indicators. We then used the 2016 BackBlaze data to evaluate its performance. Our results show that our decision tree identifies at least 52 percent of all disk failures and makes nearly all its predictions several days ahead: no more than 2.45 percent of the predicted failures occur within one day or two of the prediction.

Finally, we compared the performance of our predictor with those of the RAIDShield and the original BackBlaze predictor. We found out that RAIDShield could predict at most 18 percent of disk failures, that is, 34 percent fewer failures than our decision tree while the BackBlaze predictor predicted 60 percent of disk failures but generated 4 to 5 false alarms per correct prediction.

Keywords: hard disk failure prediction, SMART attributes, machine learning, data-driven simulation

I. Introduction

Magnetic disk drives offer the most cost-effective way to store large amounts of data. At the same time, they happen to be the least reliable component of modern computers, mostly because they include fast moving parts [1], [2]. As a result, all disk-based storage systems must include provisions for preventing data losses, such as replication and omission correcting codes [3], [4].

A more proactive approach can supplement these provisions. Rather than waiting for disks to fail, we could try to predict which ones are the most likely to fail by periodically sampling their SMART attributes, several of which are positively correlated with an increased risk of disk failure. This would let us save ahead of time the contents of the suspected disks, thus reducing the risk of data loss. Several recent studies show the viability of the approach [5], [6], [7]; their sole limitation resides in their focus on homogeneous populations of disks from the same make and model. In reality, many data centers include disks from multiple makes and models as new disks are constantly added to replace failed ones or to expand their storage capacity. One well-known instance is the BackBlaze disk farm [8]. To address this issue, we decided to focus our study on an exemplar of a heterogeneous disk population, namely, that described in BackBlaze quarterly hard drive reliability reports [8]. We used the 2015 data to train our predictors and data from 2016 to evaluate their performance, treating all disks equally regardless of their specific makes and models. Given the high number of SMART attributes that were missing for many disks, we used a two-step approach. First, we identified six SMART attributes that were the most correlated with an impending disk failure for our whole disk population. Second, we use machine learning to build a decision tree discrimination between true failure predictions and false alarms among the small minority of disks that tested positive for at least one of our failure indicators.

The results presented here illustrate the limits and the benefits of the approach. We found out that we could predict around 52 percent of all disk failures. The main culprit is the lack of standardization among the various SMART attributes that are reported by disks of different makes, which greatly reduced the number of attributes that were good overall failure indicators for our disk population.

We also compared the performance of our predictor with those of the RAIDShield [9] and the original BackBlaze predictor. We found out that RAIDShield could predict at most 18 percent of disk failures, that is, 34 percent fewer failures than our decision tree. In addition, RAIDShield could only achieve this result by setting its detection threshold at the lowest possible level, producing many more false alarms than correct predictions. The BackBlaze predictor would predict 60 percent of disk failures, that is, 8 percent more than our predictor but generated four to five false alarms per correct prediction.

II. Related Works

Xu *et al.* [5] used a health status assessment parameter (instead of a binary parameter) based on the SMART values to introduce a novel method based on recurrent neural networks to predict disk failures. In 2014, researchers from the same group (Li [7]) used classification and regression trees to predict disk failures based on the same health status assessment parameter. Both works applied the same methodology based on the reverse

SummerSim-SPECTS, 2017 July 9-12, Bellevue, Washington, USA;
©2017 Society for Modeling & Simulation International (SCS)

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

arrangement test, rank-sum test, and z-scores to select the SMART parameters. They also used the same three datasets, each consisting of disks from the same manufacturer, to train, validate, and score their models. The results from both papers showed at least 95 percent accuracy for the best test case. However, due to the homogeneity of their datasets and the missing information about the number of models per dataset, their solutions will not work with high accuracy for data centers with disks from other manufacturers.

Botezatu *et al.* [6] presented a method based on decision trees to predict the necessity of a disk replacement based on the SMART parameter values. This approach utilized statistical techniques to automatically select the SMART parameters that correlate with a disk replacement. The results showed a 98 percent accuracy using as dataset 30,000 disks from two manufacturers (monitored over a period of 17 months). The proposed solution will not be applicable to a real-world data center because the researchers only used a limited dataset (one model per manufacturer for the training and validation process, and one model per manufacturer for the scoring process) to train, validate, and score the proposed model.

Zhu *et al.* [10] applied support vector machines and neural networks to predict disk failures. They used a dataset from a real data center consisting of 23,395 identical model drives. The results showed that the highest failure detection rate (95 percent) was achieved with the neural network model while the lowest false alarm ratio (0.03 percent) was achieved with the support vector machine model. Even though the authors used a real-world dataset, the distribution of the disks in the dataset (only one model) makes these results not applicable to heterogeneous environments.

Other studies, among them [11], [12], [13], and [14], applied different machine learning and statistical solutions to predict disk failures based on the SMART parameter values. In all four cases, their authors only considered population consisting of the same make and often the same models.

More recently, Jing Li *et al.* [15] proposed to measure the effectiveness of disk failure prediction models using the migration rate and the mismigration rate that result from the model predictions. Their work shows that warning time is as important as the accuracy of the model when measuring the performance of disk failure predictor.

We propose here to develop a different methodology to train, validate and score a disk failure prediction model by using a real-world dataset that includes disks from different manufacturers and models. This should help us understand the limitations of the selected machine learning techniques to predict disk failures in such environments.

III. Disk Failure Prediction

A. The Datasets

To work with a real-world dataset from a heterogeneous environment, we decided to use data from BackBlaze [8]. They measure 88 SMART parameters (44 raw and 44 normalized) per disk per day. To train and validate our model, we use the data from 2015 (17,509,251 records), while for scoring our model we use the data from 2016 (24,472,345 records).

The 2015 dataset from BackBlaze includes 62,898 drives. The fraction of drives that failed in this period is 2.27 percent (1,428 drives). Table I shows the distribution in terms of the manufacturer, the number of drives, and the number of models for the 2015 dataset.

TABLE I: BackBlaze 2015 Dataset Distribution

Manufacturer	Number of Disks	Number of Models
HGST (Western Digital)	10,384	5
Hitachi	12,991	9
Samsung	2	1
Seagate	36,213	24
Toshiba	247	3
Western Digital	3,061	36
Total	**62,898**	

The 2016 dataset from BackBlaze consists of 81,173 drives. The fraction of drives that failed in this period is 1.76 percent (1,431 drives). Table II shows the distribution in terms of the manufacturer, the number of drives, and the number of models for the 2016 dataset.

TABLE II: BackBlaze 2016 Dataset Distribution

Manufacturer	Number of Disks	Number of Models
HGST (Western Digital)	17,383	5
Hitachi	12,727	9
Samsung	1	1
Seagate	47,947	21
Toshiba	339	4
Western Digital	2,776	37
Total	**81,173**	

B. SMART Parameters Selection

To select the SMART parameters to build our model, we decided to use a methodology based on the following criteria:

a) *SMART parameter type (Raw and Normalized)*: Each attribute has a raw value, whose measurement is entirely up to the drive manufacturer (counts or a physical unit, such as degrees Celsius or seconds), and a normalized value, that transforms the raw value using a scale from 0 (bad) to some maximum (good) value. The maximum value is manufacturer and model dependent (different models from the same manufacturer can have a different initial value = 100, 200, or 253). We decided to use the RAW values of the selected parameters in our study because they were the most likely to remain the same among various makes and models.

b) *Statistical Analysis*: To keep the number of inputs to a manageable size and to eliminate irrelevant variables (parameters without any relationship to the target), we performed a statistical analysis of the 44 RAW parameters from the dataset using SAS Enterprise Miner. We applied the R-Square and Chi-Square methods to rank these 44 parameters based on their fitness for predicting disk failures while rejecting those with more than 90 percent of missing values. After this procedure, we were able to reduce the number of SMART parameters in our dataset to 21 variables by considering only the relevant parameters present in at least 10 percent of the disks in the studied dataset.

c) *Trend test*: We decided to perform a reverse arrangement test on the 21 SMART attributes selected from BackBlaze

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

dataset after applying the statistical analysis. We calculated the percentage of good and failed drives that have a trend per SMART parameter.

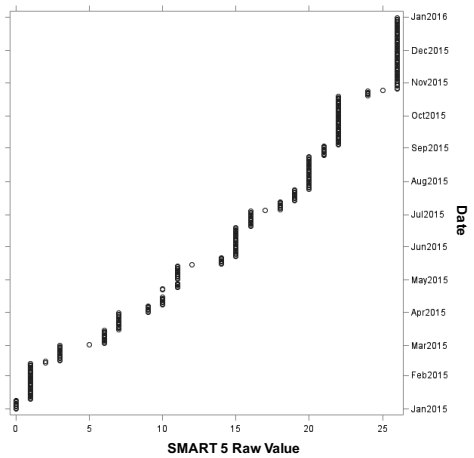


Fig. 1: Disk with a Trending Parameter

Figure 1 shows an example of a disk with a positive trending parameter (SMART 5 RAW), where the value of the studied variable increases as the time progresses.

Table III shows the results of the trend test. Based on these results, we selected the following parameters using the percentage of disks with a trend (high percentage of failed disk with a trend and low percentage of good disks with a trend):

- Reallocated Sectors Count (SMART 5): is the number of the bad sectors that have been found and remapped.
- Reported Uncorrectable Errors (SMART 187): is the number of errors that could not be recovered using hardware ECC.
- Command Timeout (SMART 188): is the number of aborted operations due to HDD timeout.
- Reallocation Event Count (SMART 196): is the number of attempts to transfer data from reallocated sectors to a spare area.
- Current Pending Sector Count (SMART 197): is the number of sectors waiting to be remapped because of unrecoverable read errors.
- Uncorrectable Sector Count (SMART 198): is the number of uncorrectable errors when reading/writing a sector.

Finally, it is important to mention that our six attributes include the five attributes used by BackBlaze to predict the failure or potential failure of a disk, showing that our trend test analysis agrees with the practical knowledge applied by BackBlaze to solve the hard disk failure detection problem.

C. Dataset Sampling

As we have seen, the 2015 BackBlaze dataset contained an overwhelming majority of disks that did not fail and a tiny minority of bad disks. Such unbalances make it harder for any machine learning technique to produce a predictor that could not neglect the failed disks.

Using our knowledge of the domain, we decided to solve this problem by using an informed undersampling method. We used the one-sided selection (OSS) method [16], to select a representative subset of the majority class E and combines it with the set of all minority instances $S_{\min}$ to form a preliminary set N, N=$E \cup S_{\min}$.

TABLE III: Reverse Arrangement Test SMART Parameter Distribution

Parameter	Description	%Good	%Bad	% Disks w/o Missing Values
Smart_5_Raw	Reallocated Sectors Count	1.887243	28.9916	100
Smart_187_Raw	Reported Uncorrectable Errors	1.562193	38.2227	56.69
Smart_188_Raw	Command Timeout	1.247511	10.27837	56.97
Smart_196_Raw	Reallocation Event Count	1.584494	27.32794	43.30
Smart_197_Raw	Current Pending Sector Count	1.417423	38.02521	100
Smart_198_Raw	Uncorrectable Sector Count	0.977863	27.10084	100

To implement the selected unbalanced dataset solution, we define $S_{\min}$ (minority class) as all instances of bad disks with at least one of the selected parameters with a trend, and E (majority class) as all instances of good disks with at least one of the selected parameters with a trend. Based on these definitions, we present the distribution of the instances for $S_{\min}$ and E:

Minority Class (Bad Disks):

- Total = 1,428.
- Bad disks with at least one parameter different than zero or missing = 891 (62.40 percent).
- Bad disks with all selected parameters equal to zero or missing = 537 (37.60 percent).

Majority Class (Good Disks):

- Total = 61,470.
- Good disks with at least one parameter different than zero or missing = 3,133 (5.09 percent).
- Good disks with all selected parameters equal to zero or missing = 58,337 (94.91 percent).

Our goal is to build the decision boundary between the minority class (failed disks) and the majority class (good disks), by only using 6.37 percent of the instances. The sole drawback of the approach is that it will classify as good all drives that have all six parameters equal to zero or missing.

D. Machine Learning Technique Selection

We implemented three different machine learning models (Decision Trees, Neural Networks, and Logistic Regression) using SAS Enterprise Miner version 14.1.

For the decision tree model, we used the following parameters:

- Max number of branches = 2.
- Max depth = 10.
- Assessment measure = Misclassification rate.

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

- (SAS automatically trains and prunes the tree based on the training and validation dataset).

For the neural network model, we used the following parameters:

- Architecture = Block Layers.
- Max number of interactions = 50.
- Max number of hidden units = 100.
- Activation functions = Direct, Logistic, Sine, Softmax, and Tanh.
- (SAS automatically selects the best network based on the selected parameters).

For the regression model, we used the following parameters:

- Main Effects.
- Two-factor Interactions.
- Polynomial terms (polynomial degree = 3).
- Regression type = Logistic.
- (SAS automatically selects the best regression model based on these parameters).

We decided to modify the prior probabilities of our training dataset to improve the capability of the proposed model to detect disk failures. The original training proportions were 0.22142 for the failed disks and 0.77858 for the good disks, and the new decision prior probabilities used are 0.5 for the minority class and 0.5 for the majority class. To validate our models, we used a five repetition ten-fold cross validation to provide ten random partitions of the original sample per repetition.

Model Comparison:

Tables IV and V present the results for the Fit Statistics and Confusion Matrix for the selected machine learning methods:

TABLE IV: Fit Statistics for the Selected Machine Learning Methods

Model	Misclassification Rate	Average Squared Error	ROC Index	Gini Coefficient
Decision Tree	0.15805	0.10036	0.883	0.766
Regression	0.22142	0.22142	0.5	0
Neural Network	0.22145	0.22145	0.5	0

TABLE V: Confusion Matrix for the Selected Machine Learning Methods

Model	False Negative	True Negative	False Positive	True Positive
Decision Tree	110	2,607	526	781
Regression	891	3,133	0	0
Neural Network	891	3,133	0	0

Out of these three methods, we selected the decision tree because it has the lowest misclassification rate (15.80 percent) and the highest number of true positives (781 failed disk predicted). The poor performance of both neural network and regression is a consequence of the presence of too many missing values for all instances of the minority class (disk failures). We then tried to impute the missing values using different parameters (median, mean, min, and max). Even though we saw an improvement for both neural network and regression after replacing the missing values with the median, the model with the best performance still was the decision tree.

Figure 2 shows the decision tree generated by SAS Enterprise Miner based on the selected parameters. For each node, we have the node id, the percentage of good disks (labeled as 0), the percentage of failed disks (labeled as 1), and the number of drives.

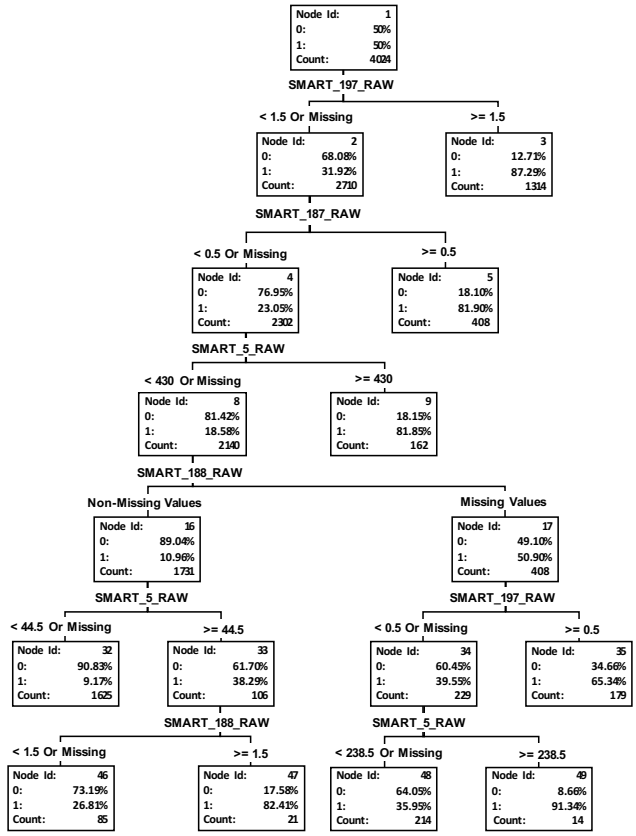


Fig. 2: Disk Failure Prediction Model based on Decision Trees

The limitation of the proposed model is shown by node 17. This node includes all instances with a zero or missing value for the SMART parameters 197, 187, 5 and 188, with a distribution of 49.10 percent of failed disks and 50.90 percent of good disks. This result shows that our model performs worst when an instance has all parameters equal to zero or missing.

IV. Experimental Results

We present the results of our study of the proposed model by using the 2015 and 2016 datasets from BackBlaze. We analyzed the failure detection rate (FDR), taking into consideration the fraction of predictable failures by using only the failed disks with at least one parameter greater than zero ($S_{\min}$ subset).

A. Testing with the complete 2015 Dataset

Since we built our model only 6.37 percent of the 2015 BackBlaze dataset, we had to check first how our model would perform on the whole 2015 BackBlaze dataset.

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

TABLE VI: Model Validation Results using the BackBlaze 2015 dataset

Failures Predicted	Failures Missed	False Alarms	Good Predicted
778	650	526	60,944

As we can see in table VI, our model achieves a failure detection rate (FDR) of 54.48 percent. To understand the behavior of the FDR for our model, we calculate the fraction of failures with non-zero values for one of our six indicators. From 1,428 failed disks in the dataset, only 891 have non-zero values, therefore the fraction of predictable failures is equal to 87.31 percent. This result shows that our model was able to predict almost 88 percent of the failures used to build the decision tree. For the false alarm rate (FAR), our model shows a value of 0.85 percent (because the number of true negatives is high).

B. Scoring with the 2016 BackBlaze Dataset

For the actual evaluation of the performance of our model, we used the 2016 dataset from BackBlaze.

TABLE VII: BackBlaze 2016 Scoring Results

Failures Predicted	Failures Missed	False Alarms	Good Predicted
747	684	551	79,191

As we can see in table VII, our model shows a failure detection rate (FDR) of 52.20 percent. For the studied dataset, only 858 failed disks have non-zero values. Therefore, the fraction of predictable failures is equal to 87.06 percent. This value can be explained by the relatively high number, 573 out of 1,431, that were not accompanied by any change in the values of any of our six SMART parameters. As our method was able to predict 747 failures out of the remaining 858 failures, it indeed predicted 87 percent of all failures that it could predict. For the false alarm rate (FAR), our model shows a value of 0.69 percent.

The failure detection rate (FDR) and the false alarm rate (FAR) are closely related to both predictable and unpredictable failures. Figure 3 shows the relationship between these parameters for the datasets.

To evaluate the timeliness of our predictions, we collected failure latencies, that is, the time interval between each failure prediction and the actual failure. A latency shorter than one or two days would give not enough time to save the information from a failed drive. Conversely, predicting disk failures too far ahead of time will offer little useful guidance to the system administrator.

Figure 4 shows the distribution of the predictions in terms of days. The results show that only 2.40 percent of the predictions occur within two days of the actual failure.

We also measured the latency in weeks to show how much ahead of time our model is making the predictions. Figure 5 shows the distribution of the predictions in terms of weeks.

These results show that 72.55 percent of disk failures are detected within four weeks of the actual failure date.

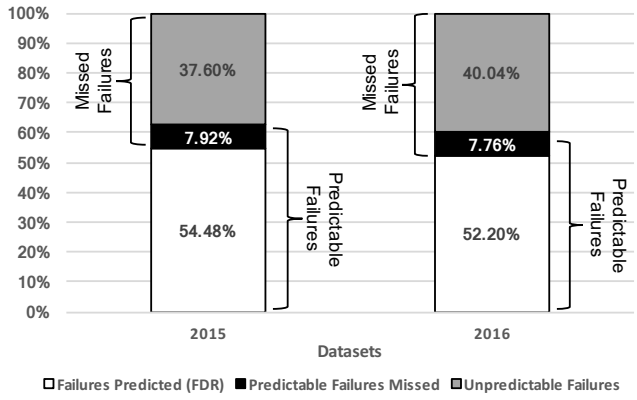


Fig. 3: Relationship between FDR and FAR

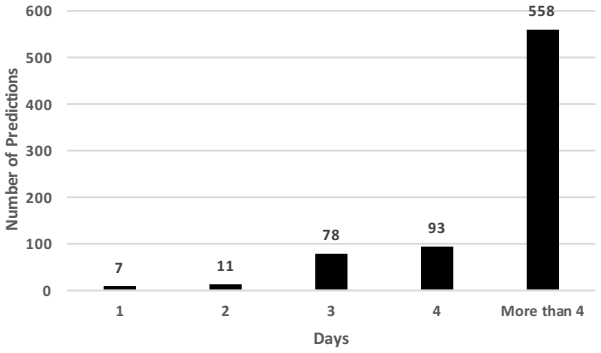


Fig. 4: Prediction Latency (Days)

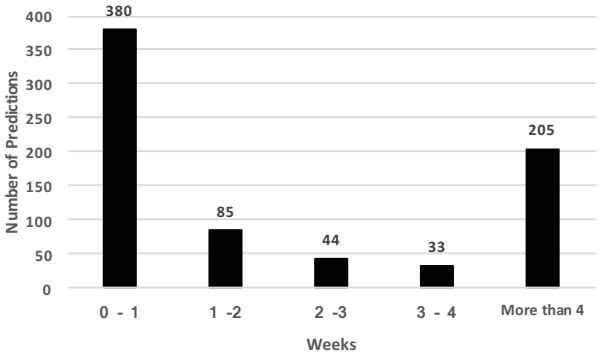


Fig. 5: Prediction Latency (Weeks)

C. Performance Comparison

We decided to compare the performance of our predictor against two extant predictors:

1) The disk failure predictor used in RAIDShield [9], which assumes that an excessive amount of sector errors typically precedes the failure of the whole-disk.
2) A baseline model used by BackBlaze that predicts a

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

disk failure each time any of the five relevant SMART parameters turns positive.

RAIDShield:

RAIDShield uses the SMART 5 Raw parameter (Reallocated Sector Count) to predict disks that are prone to failures. If the SMART 5 Raw exceeds a given failure threshold, the disk is considered to have become unreliable. The dataset they used to evaluate the performance of their detection mechanism is based on a population of 100,000 disks from the same family. Their results show a failure detection rate between 52 and 70 percent and a false alarm rate between 0.8 and 4.5 percent for SMART 5 values between 20 and 200.

We measure the performance of the RAIDShield predictor [9] using the 2016 dataset from BackBlaze. We used a range of values for the SMART 5 parameter between 1 and 600. We then compare these results against the results of our predictor for the same dataset.

Figure 6 shows the failure detection rate comparison between our predictor and RAIDShield detection mechanism. The results show that our predictor significantly outperforms RAIDShield detection mechanism with a difference between 36.82 percent (for SMART 5 value = 1) and 48.98 percent (for SMART 5 value = 600).

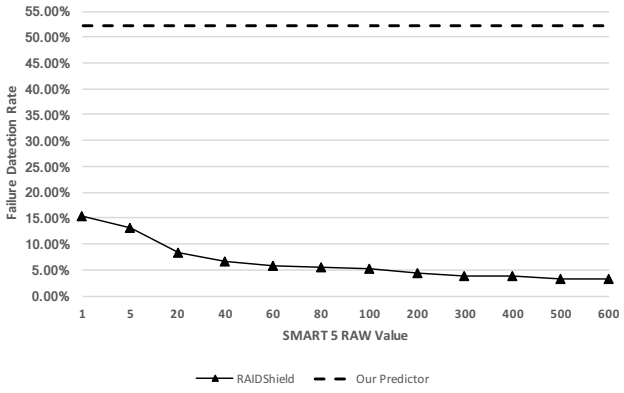


Fig. 6: Failure Detection Rate Comparison (Our Predictor vs RAIDShield)

Figure 7 shows the false alarm rate comparison between our predictor and RAIDShield detection mechanism. The very low FAR achieved by RAIDShield can be simply explained by observing that it makes much fewer predictions, good or bad, than our predictor.

These results are much worse than those reported by the authors of RAIDShield. This should not surprise us because RAIDShield was originally developed for and tested on a homogeneous population of disks, all the same make and model. In our study, only 18 percent of the failing disks were found to display any changes in the value of the SMART attribute associated with the number of relocated sectors before failing.

We conclude that our solution improves significantly the prediction of disk failures in heterogeneous environments when compared to RAIDShield while offering a similar performance for the true to false alarm ratio.

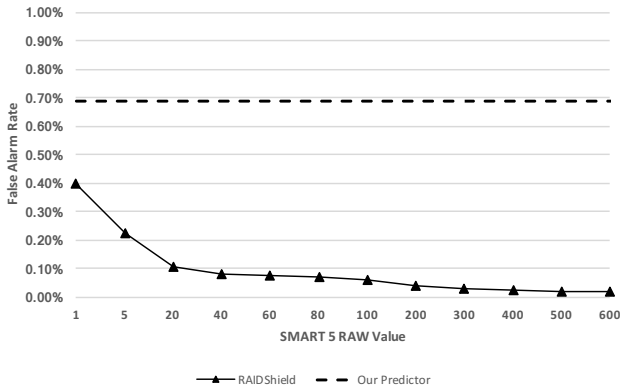


Fig. 7: False Alarm Rate Comparison (Our Predictor vs RAIDShield)

BackBlaze Predictor:

The original BackBlaze predictor uses five SMART parameters (SMART 5, SMART 187, SMART 188, SMART 197, and SMART 198). In other words, they use five out of our six parameters, missing SMART 196. They verify the status of a disk when the RAW value for one of these five attributes is greater than zero.

Since the baseline model predicts a disk failure each time any of our failure indicators turns positive, we can expect it to have a higher failure detection rate than our predictor. This was indeed the case: when we applied to the 2016 data, the baseline model correctly predicted 858 of the 1,431 observed disk failures, thus achieving a 60 percent failure detection rate, which is better than our predictor. This better performance came however with a price: the model issued 3,916 false alarms, that is, slightly more than 4.5 false alarm per correct prediction. Table VIII shows the results of the scoring process for the 2016 BackBlaze dataset with the baseline model:

TABLE VIII: Scoring the Baseline model with BackBlaze 2016 dataset

Failures Predicted	Failures Missed	False Alarms	Good Predicted
858	573	3,916	75,826

In addition, we observed that 25 percent of its predictions were made more than sixty days ahead of time of the actual disk failure. This very long delay would greatly complicate the task of asserting the correctness of the predictions made by the model. It could indeed result in having many of these long-term predictions incorrectly classified as false alarms. Should this be the case, the effective failure prediction rate of the model would fall below the failure detection rate of our predictor.

V. Conclusion

We have presented a decision-tree based disk failure predictor for heterogeneous populations of disks, such as the ones

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

encountered in many large data centers. The main problem we encountered was the lack of standardization among the SMART attributes of various makes and models of the disk population we investigated. As a result, we were only able to identify six SMART attributes that were the strongly correlated with an impending disk failure for our whole disk population. We found out that only a small minority of disks (around 3 percent) were flagged by one or more of these attributes. We then used machine learning to build a decision tree that would separate true failure predictions from false alarms among the flagged disks. Our decision tree was built using disk reliability data collected at BackBlaze in 2015 and scored using data collected there during 2016.

Our results indicate that we can predict 52 percent of all disk failures. This corresponded to 87 percent of the disk failures that were preceded by any change in one of this SMART attributes we monitored.

We also compared the performance of our predictor with that of the RAIDShield predictor and that of a baseline prediction model that predicted a failure each time one of our failure indicators turned positive. We found out that RAIDShield performed very poorly in our heterogeneous environment and was never able to predict more than 18 percent of disk failures. While the BackBlaze predictor (baseline model) could predict 60 percent of disk failures, it also generated between 4 and 5 false alarms per correct prediction. These results illustrate the difficulty of developing a good generic disk failure predictor for a heterogeneous population of disks.

In a future work, we plan to regroup disks by manufacturer and construct a separate disk failure predictor for each make. This should allow us to include more SMART attributes in our decision trees and, hopefully, result in higher failure detection rates. Even then, our generic predictor would still apply to makes of disks that are barely present in the disk population or were recently inserted.

REFERENCES

[1] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, (Berkeley, CA, USA), pp. 17–29, USENIX Association, 2007.

[2] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you?," in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, FAST '07, (Berkeley, CA, USA), pp. 1–16, USENIX Association, 2007.

[3] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, SIGMOD '88, (New York, NY, USA), pp. 109–116, ACM, 1988.

[4] T. J. E. Schwarz and W. A. Burkhard, "RAID organization and performance," in *Proceedings of the 12th International Conference on Distributed Computing Systems*, pp. 318–325, Jun 1992.

[5] C. Xu, G. Wang, X. Liu, D. Guo, and T. Y. Liu, "Health status assessment and failure prediction for hard drives with recurrent neural networks," *IEEE Transactions on Computers*, vol. 65, pp. 3502–3508, Nov 2016.

[6] M. M. Botezatu, I. Giurgiu, J. Bogojeska, and D. Wiesmann, "Predicting disk replacement towards reliable data centers," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 39–48, ACM, 2016.

[7] J. Li, X. Ji, Y. Jia, B. Zhu, G. Wang, Z. Li, and X. Liu, "Hard drive failure prediction using classification and regression trees," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 383–394, June 2014.

[8] "BackBlaze quarterly hard drive reliability reports." https://www.backblaze.com/b2/hard-drive-test-data.html. Accessed: 2016-11-20.

[9] A. Ma, F. Douglis, G. Lu, D. Sawyer, S. Chandra, and W. Hsu, "RAIDShield: Characterizing, monitoring, and proactively protecting against disk failures," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, (Santa Clara, CA), pp. 241–256, USENIX Association, 2015.

[10] B. Zhu, G. Wang, X. Liu, D. Hu, S. Lin, and J. Ma, "Proactive drive failure prediction for large scale storage systems," in *2013 IEEE 29th Symposium on Mass Storage Systems and Technologies (MSST)*, pp. 1–5, May 2013.

[11] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *ACM Comput. Surv.*, vol. 42, pp. 10:1–10:42, Mar. 2010.

[12] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *J. Mach. Learn. Res.*, vol. 6, pp. 783–816, Dec. 2005.

[13] J. F. Murray, G. F. Hughes, and K. Kreutz-Delgado, "Hard drive failure prediction using non-parametric statistical methods," in *Proceedings of ICANN/ICONIP*, 2003.

[14] G. Hamerly and C. Elkan, "Bayesian approaches to failure prediction for disk drives," in *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, (San Francisco, CA, USA), pp. 202–209, Morgan Kaufmann Publishers Inc., 2001.

[15] J. Li, R. J. Stones, G. Wang, Z. Li, X. Liu, and K. Xiao, "Being accurate is not enough: New metrics for disk failure prediction," in *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pp. 71–80, Sept 2016.

[16] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, pp. 1263–1284, Sept. 2009.

Authorized licensed use limited to: Univ of Calif Santa Cruz. Downloaded on August 25,2020 at 00:26:52 UTC from IEEE Xplore. Restrictions apply.

UC Santa Cruz

UC Santa Cruz Previously Published Works

Title

RESAR: Reliable Storage at Exabyte Scale

Permalink

https://escholarship.org/uc/item/10p5q5v4

Authors

Schwarz, SJ Thomas
Amer, Ahmed
Kroeger, Thomas
et al.

Publication Date

2016

DOI

10.1109/mascots.2016.13

Peer reviewed

Darell Long, Ph.D.
09/17/2025
Exhibit K
H. Bautista, CSR #11600

eScholarship.org Powered by the California Digital Library
University of California

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 385 of 411

RESAR: Reliable Storage at Exabyte Scale

Thomas Schwarz, SJ*, Ahmed Amer †, Thomas Kroeger‡, Darrell Long§, Jehan-François Pâris¶
* Universidad Centroamericana, La Libertad, El Salvador, Email: tschwarz@uca.edu.sv
†Santa Clara University, Santa Clara, CA, Email: aamer@scu.edu
‡Sandia National Laboratories, Livermore, CA,[1] Email: tmkroeg@sandia.gov
§SSRC, University of California, Santa Cruz, CA, Email: darrell@soe.ucsc.edu
¶University of Houston, Houston, TX, Email: jfparis@uh.edu

***Abstract*—Large-scale disk-based storage systems need to protect the data stored in them against individual disk failures, common component failures and latent disk errors. We present RESAR, a layout scheme that provides two failure tolerance by only using XOR operations to calculate parity data. Our layout has the same write overhead as that of a disk array whose layout is based on virtual RAID Level 6 disk arrays. If the size of a reliability stripe is k, our write overhead is $2/k$. We show that RESAR is actually more resilient than the RAID Level 6 layout.**

I. Introduction

In the age of big data, magnetic disk based storage is expanding, but also slowly moving to a more archival role; magnetic hard drives are replaced by solid state disks and more data is stored but proportionally less of it is accessed. These trends imply a continuous increase in the capacity and economic life span of large disk based storage systems. In order to save energy, most disks in such a large storage system will be powered off. The large number of components has made failures a daily occurrence. The value of data will determine the level of failure protection, but most storage systems will do fine with tolerance of two simultaneous failures, if the failures can be repaired very quickly. The need for two-failure tolerance arises from the presence of latent sector errors that can only be detected by frequent scrubbing or by reading an affected sector, which all too often happens during a repair. A repair here is the reconstruction of data stored originally in a lost disk and moved to different, live disks and is separate from a replacement of the physical drive, which takes place at a later time or is subsumed by adding many more drives to the storage system.

Traffic in a large storage center is dominated by writes with reads to random locations. It is best therefore to organize the data in the form of one or several log and store the incoming data on a few disks that are currently powered on. The use of disks with shingled write recording is another reason to store data in form of a log.

We can achieve two-failures tolerance using erasure correcting codes such as Row-Diagonal Parity or Reed-Solomon codes, but we present here an alternative that is using a flat XOR-code. As usual, we decluster, that is, we store data in relatively small "disklets" (of between 10GB and 50GB with today's disk capacities) that can be read and written within minutes. This means recovery times in the order of minutes versus hours for a standard RAID Level 6. If disk failures are detected fast, this narrows the "window of vulnerability" that opens when a disk fails and subsequent failures can lead to dataloss.

Our family of layouts called RESAR places each *data disklets* (so called because it contains user data) into two reliability stripes and adds to each reliability stripe an additional *parity disklet* that contains the exclusive-or of the data in the data disklets disklets in the stripe.

By introducing a disklet layer into the organization of a disk array, we need a metadata server that manages the mapping of disklets to disks, but obtain the following advantages
(1) We can deal with very large disk arrays consisting of heterogeneous disks.
(2) Only the metadata server has to deal with relocating data in order to balance the load between already present disks and a batch of new disks.
(3) By making it simple to store data in a log, RESAR allows to minimize the number of disks currently powered-on for writing.
(4) RESAR allows to change effectively the size of reliability stripes, allowing to make dynamic trade-offs between parity overhead and the effort needed to recover from disk failures.
(5) RESAR recovery from disk failure is in the order of minutes (compared to hours for a system composed of classical RAID Level 6 layouts).
(6) RESAR allows to recover using two distinct set of disks and is thus much less likely to interfere with an important read operation.

In the remainder of the article, we explain the way to create RESAR layouts and assess RESAR's reliability by comparing to a similar disk array that is also declustered and organizes disklets in reliability stripes of k disklets and two additional parity disklets. The presence of the second parity disklet in the stripe then requires the use of a more sophisticated code (such as generalized Reed-Solomon with Galois field calculation, Even-Odd, or Row-Diagonal Parity). Because the stripes are built as in a RAID Level 6 layout, we refer to this organization in short-hand as a RAID Level 6, though it should be properly called a declustered RAID Level 6.

[1] Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 386 of 411

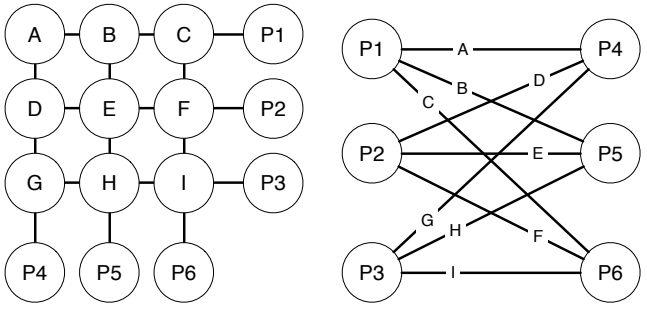


Fig. 1. Left: The two-dimensional layout for a disk array. A, B, ... I are data disks and $P1$, ..., $P6$ parity disks. Right: The corresponding graph visualization.

II. Two-Failure Tolerant Flat XOR Codes

Flat XOR codes are defined by Greenan *et alii* to be codes where parity symbols are calculated from certain subsets of data symbols [1]. We call these subsets reliability stripes. To each flat XOR-code corresponds directly a disk array layout where each data symbol corresponds to a data disk and each parity symbol to a parity disk.

Layouts based on flat XOR-codes and that tolerate two simultaneous failures need to place each data disk in at least two different reliability stripes. The intersection of two reliability stripes cannot contain more than a single disklet. We can *label* each data disk by the numbers of the two reliability stripes to which it belongs. Similarly, we can label each parity disk with the number of the reliability stripe to which it belongs. This defines a graph structure derived from the disk layout where each parity disk corresponds to a vertex and each data disk to an edge between vertices.

We use the graph to visualize the layout of disklets into reliability stripes. Figure 1 gives an example. In the graph layout, parity disklets corresponds to vertices and data disklets to edges. To our knowledge, the graph visualization was first exploited by Xu and colleagues in the definition of B-codes [4]. An observation by Zhou and colleagues characterizes minimal failure sets of disks as those containing either a cycle of edges or a path where the end vertices have also failed [5]. The graph visualization is a good way to determine the failure resilience of these type of layouts, as we did in previous work for the "complete layout" so-called because it derives from a complete graph. [3].

We investigate here a RESAR layout based on the bipartite graph, but allowing us to assign a parity layout for an arbitrary number of data disklets. The layout consists of two columns of parity disklets (the P and the D-disklets). Each P-parity disklet is "connected" (by a data disklet) to its opposite D-disklet and the next $k-1$ D-disklets. If we so desire, we can have the layout loop around so that all reliability stripes encompass exactly k data disklets.

We number all disklets in a RESAR layout by a pair of numbers (i, j) where $0 \leq j \leq k+1$ and i enumerates consecutively the pairs of P- and D-parities. The P-parity gets $i = 0$ and the D-parity gets $i = 1$. The data disklets are

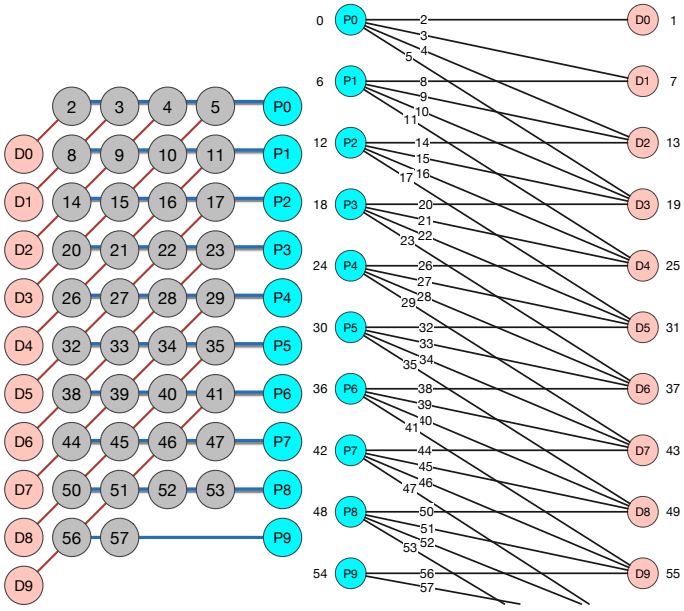


Fig. 2. A small bipartite RESAR layout (left) and its graph representation with $k = 3$. The disks are numbered for rack-aware layout (Section XXX)

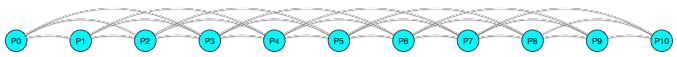


Fig. 3. A small complete RESAR layout in graph form.

numbered in order. We then convert the pair of numbers to a single one by the formula $(i, j) \rightarrow i * (k+2) + j$ that gives an enumeration of all disklets in the design such that neighboring disklets are assigned close indices. Since failure patterns with data loss are composed of neighboring disklets, we can use enumeration in order to insure that a single disk or rack failure cannot generate data loss. In our scheme, if the indices of two disklets is apart by more than $k^2 + k$, then they cannot be neighbors.

An alternative layout (the complete RESAR layout) is based on "stretching out" a complete graph and is presented in Figure 3. There, we have a single line of parity disklets (= graph vertices) and connect each disklet to the next k parity disklets down the line. Since each parity disklet also receives data disklets (= edges) from its left neighbors, each vertex (= parity disklet) in the graph is connected to $2k$ edges, which means that each reliability stripe is made up of $2k$ data disklets and the corresponding parity disklet. We show this layout only to make the point that RESAR defines families of disk array designs.

III. Recovery from Disk Failure in RESAR

To recover the data from a failed drive we must recover each of the disklets in the drive. Since each data disklet is in two reliability stripes (see Figure 4 for a depiction in the graph visualization), we can recover using either reliability stripe if all the other data disklet and the parity disklet in the corresponding stripe are still available. A parity disklet on a

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 387 of 411

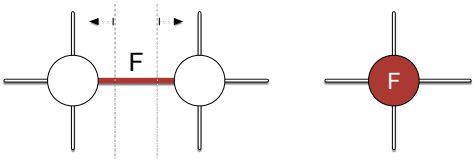


Fig. 4. Left: Failed data disklets with its neighbors. Right: Failed parity disklet with its neighbors.

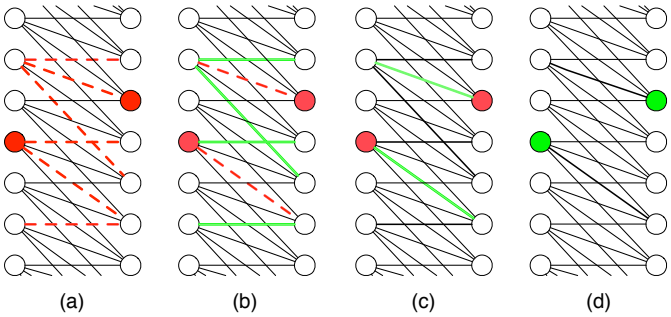


Fig. 5. Cascading recovery of multiple failures in RESAR.

failed disk can be reconstructed in another disk if all its data disklets can be recovered.

We can use the graph visualization to discuss recovery. The graph represents disklets, not disks, and a single disk failure results in multiple disklet failures. As we will discuss, a good disklet to disks mapping ensures that these disklet failures resulting from the failure of a single disk are widely spread over the (rather large) graph. We can recovery the data from a disklet (and then place the recovered disklet on another disk drive) if it is represented by an vertex (and is therefore a parity disklet) if all the edges (data disklets) adjacent to it are available. We can recover the data from a disklet represented by an edge (so this is a data disklet) if one of the adjacent vertices and all the edges adjacent to it are available. Cascading recovery in a RESAR layout

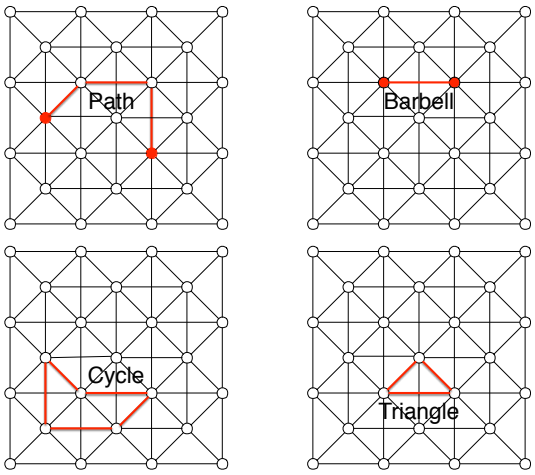


Fig. 6. Irreducible failure patterns.

happens if a disklet can only be recovered after some of its neighbors have been recovered. Figure 5 gives in column (a) a case with several failures in our graph layout. Failed elements are marked in red. In the upper failure cluster, there is one vertex on the left, which has lost three adjacent edges. For this reason, this vertex (or to be more precise, the reliability stripe represented by this vertex) cannot be used for recovery, but each data element in this stripe is also located in another reliability stripe. Two of the failed data elements can be recovered directly in the following step (column (b)), and the remaining failed element's data can be recovered in the third step (column (c)). The failed vertex in this cluster can only be reconstructed if all its adjacent edges are available. The lower failure cluster contains a failed vertex from which a path of length two of failed edges emanates. This pattern resolves itself only in the fourth step.

Arguments about failure tolerance are much easier in the graph than in the original primary design, as was previously observed [5]. Disk and sector failure induce a *failure pattern* in the graph. We are especially interested in patterns that represent data loss and that are minimal in the sense that removing one element of the pattern yields a pattern of failure from which we can recover. Any failure pattern that implies dataloss is or contains at least one minimal failure pattern. A key observation is that an edge, that is part of a minimal failure pattern, either has end-vertices that also have failed or an end-vertex where one other adjoining edge has also failed. This allows us to classify all minimal failure patterns. There are either a cycle consisting of failed edges or a path, which starts and ends at a failed vertex and otherwise consists of failed edges in between. The smallest minimal failure patterns are the *bar-bell* and the *triangle*. Figure 6 illustrates these concepts.

IV. RESAR Reliability

We now compare the reliability of a RESAR layout with that of a layout that organizes all disklets in the stripes of a RAID Level 6 layout with two parity disklets per stripe and k data disklets. The parity overhead in both organizations are equal (namely $2/k$). We also make some modeling assumptions that contradict the RESAR design goals of flexibility, but are necessary for calculations. In particular, we assume that each disk contains the same number L of disklets and that the disklets in the same position on the disk are organized in a separate RESAR layout where we connect the two ends in a single large circle. We make similar assumptions for the RAID Level 6 layout, namely that each reliability stripe contains disklets from the same relative location of the disk.

We first determine the number of failure patterns with n failed disklets out of N that cause dataloss in a single RESAR circle and in a RAID Level 6 layout, where in the latter all disklets form parts of a reliability stripe with k data and two parity disklets. We can use the notion of irreducible failure patterns (Figure 6) to determine the probability of dataloss for n failures, $n \leq 6$, before we run into an explosion of complexity in the derivation of the formulae and the formulae

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 388 of 411

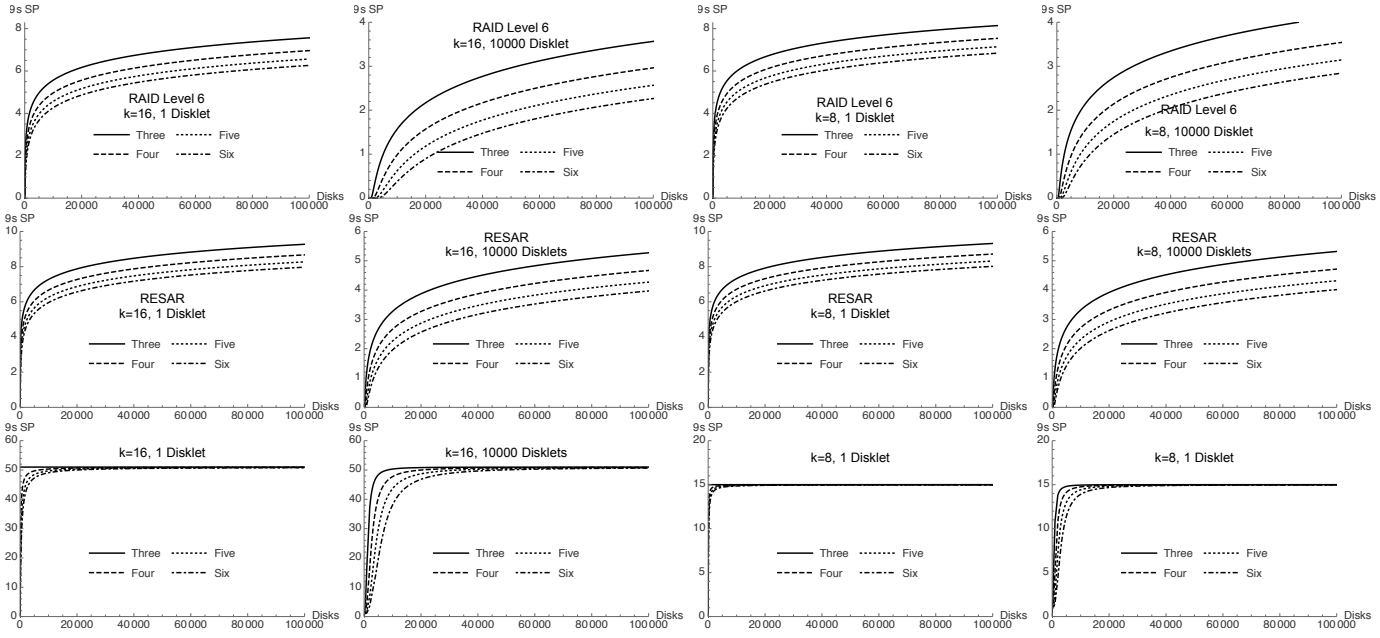


Fig. 7. Data survival probability in numbers of nine for the RAID Level 6 (top line) and the Resar layout and the increase in data survival probability by switching from the RAID Level 6 layout to the RESAR layout. Notice the different dimensions on the y-axis between the top and the middle row.

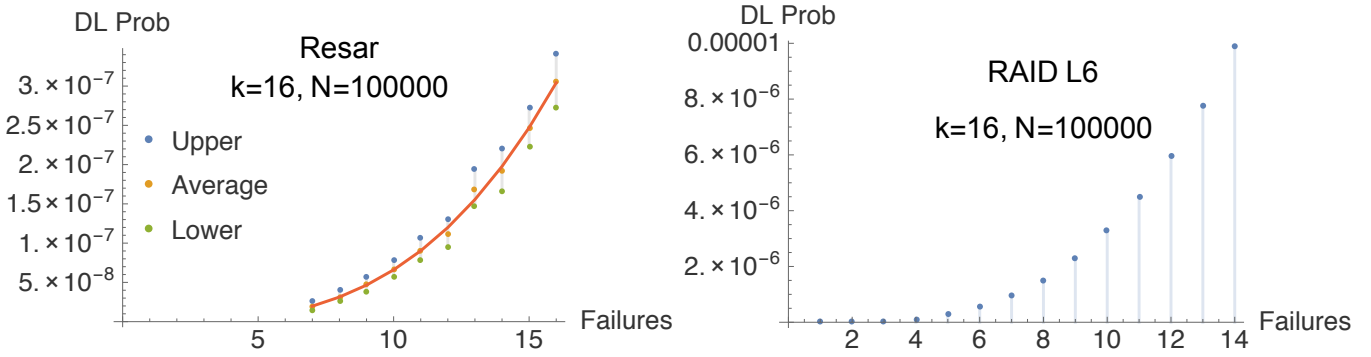


Fig. 8. Simulation results for the data loss probability of a RESAR layout (left) and exact values for a declustered RAID Level 6 layout (right), both with 100,000 disks. On the left, we give the average and the upper and lower value for the 99% confidence interval calculated with Wilson's formula.

itself. These numbers are actually quite useful for situations in which we have many disklets of few sites, since (a) recovery from dataloss is potentially so fast that the system is rarely ever in a state with more than 6 failed disks even if we ignore dataloss, and since (b) the probability of some dataloss with 7 failed disks is high (even though the amount of data lost is small). We give the results in Figure 7. As we can see, the RESAR layout is quite a bit more robust, for $k = 16$, the probability of dataloss is about 50 times higher and for $k = 8$, about 15 times higher.

Next, we simulated the resilience of a RESAR layout with 100,008 disklets (or disks and one disklet per disk). This simulation is very difficult and work-intensive (two weeks on 68 nodes) because failures are so rare. Accordingly, the results in Figure 8 (left) show great variation. We used a quadratic function for curve smoothing, giving in red.

The results show that RESAR with $k = 16$ is about fifty times less likely than an equivalent declustered RAID Level 6 layout to have suffered dataloss in the presence of the same number of failed disks. Figure 9 compares the dataloss probabilities for up to 500 failures and shows that RESAR remains considerably more robust than the declustered RAID, but at a slightly lower rate for larger numbers for failures.

Finally, Figure 10 shows the differences between the two organizations if we have many disklets per disk. As we can see, the effects are then much more pronounced. We used very small disklet sizes because even after using 60 nodes for several weeks, our simulation results have large confidence intervals that will be magnified if we calculate the robustness of layouts with many disklets per disk. $100,000$ disklets on a 10TB disk give a disklet size of 10MB, which is actually close to the block size on the new kinect drives from Seagate that use shingled write recording.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 389 of 411

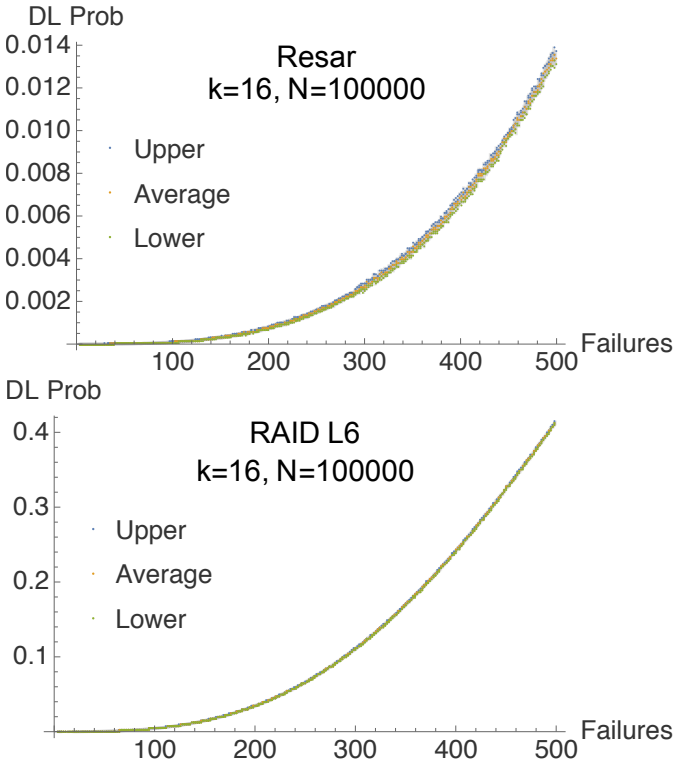


Fig. 9. Simulation results for the data loss probability of a RESAR layout (top) and a RAID Level 6 layout (bottom) with 100,000 disks. We give the average and the upper and lower value for the 99% confidence interval.

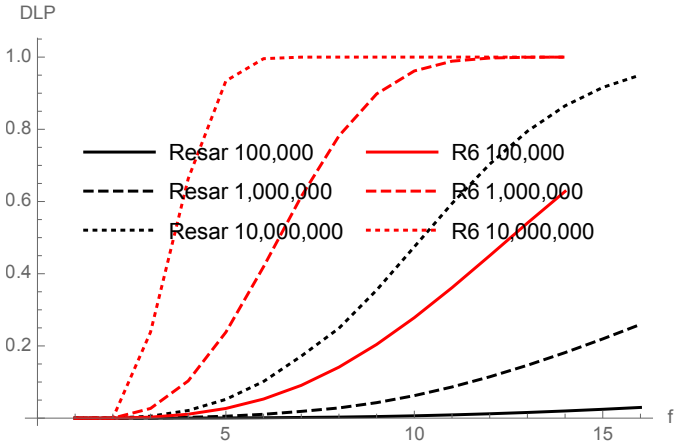


Fig. 10. Dataloss probabilities for a system with various numbers of disklets per disk.

V. Performance

RESAR is optimized for writes to consecutive data disklets. The need arises because we want to organize writes such that we write all data disklets in a reliability stripe so that we can update the parity disklet directly.

In the RAID Level 6 layout, there is only a single reliability stripe and we need to store the contents of k data disklets "in flux" before we can calculate the parities. Alternatively, we can maintain a buffer each for the two parity disklets and maintain

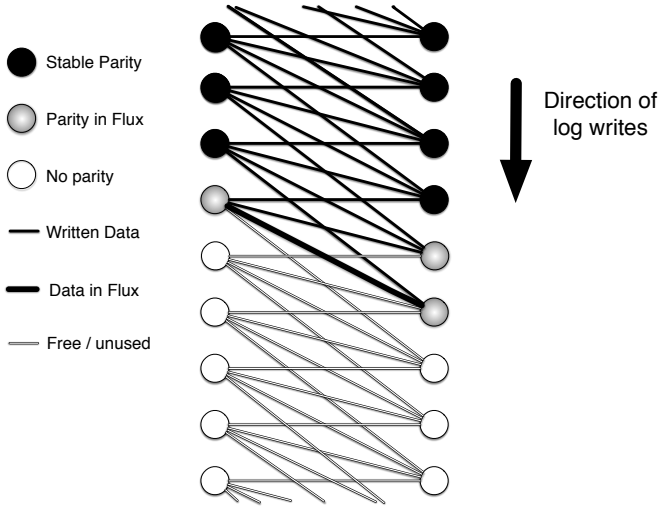


Fig. 11. Log write operation in a RESAR layout (with $k = 4$).

in it the parity of the data disklets in the stripe already written.

If we are writing a data disklet in a flat XOR-code, we can then write another data disklet in the same reliability stripe, but this leaves $k-1$ data disklets in the other reliability stripe, in which the original data disklets was, unwritten. As Figure 11 shows, the bipartite RESAR layout actually deals quite well with this problem. We write data disklets in order of their indices. A parity disklet is in flux, if some, but not all the adjoining data disklets is in flux. In Figure 11, there are three parity disklets in flux, one P-disklet and two D-disklets (the gray vertices on the right). If we write the next data disklets, one more D disklet would be in flux, but if we then move on to the next data disklet, one D disklet can be closed and no additional one enters a state of flux.

Since we are writing data disklets one at a time, only one data disklet (per log) can ever be in flux. Unlike for the RAID Level 6 layout, more than two parity disklets are in flux, namely a maximum of $k+1$.

Dealing with a large number of parity disklets in flux is in fact not difficult. The simplest mode is to store copies of data disklets contents in other disks until all data disklets in a reliability stripe have been written. At this moment, we can calculate the parity data, place it in a parity disklet and then free the disklets with the content copies. We do not need to keep the disks with the data disklets replica powered on, but only need to power them on if they are needed, first to store the replicated data, and then in order to be read for the parity calculation.

The difference between the two data layouts (RESAR and RAID Level 6) then only lies in the additional number of disklets to be stored, which is very small compared to the total capacity of disklets and costs less than the price of a single hard drive. The RAID Level 6 layout uses up to $2k$ disklets for temporary copies of data, while the RESAR layout would use up to $k(k+1)$ temporary disklets.

Most of the time, we would write to several data disklets

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 390 of 411

in parallel in order to digest the amount of incoming data. The small advantage of the RAID Level 6 layout due to its compactness would then be even smaller.

VI. Conclusions

In this paper, we have evaluated a new layout for very large disk arrays called RESAR. This layout places every disklet into two reliability groups with one parity disklet each. It has been designed to avoid small RAID writes (where we obtain the new parity from the delta of the new and the old data and the old parity). We compared it with a layout organized as a collection of RAID Level 6 reliability stripes and shown the RESAR layout to be much more resilient.

In the age of fast Galois field calculations [2], the fact that RESAR only uses exclusive-or operations to calculate parity might or might not be an advantage, but it frees us from the need to use powerful microprocessors.

We are well aware that many instances of data loss in disk arrays are due to faulty disk array controller software and that the reliability of the best layout can be destroyed for example by a faulty failure mechanism. The choice of the disk layout should not affect the complexities of implementing disk arrays.

References

[1] K. M. Greenan, X. Li, and J. J. Wylie, "Flat XOR-based erasure codes in storage systems: Constructions, efficient recovery, and tradeoffs," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–14.

[2] J. S. Plank, K. M. Greenan, and E. L. Miller, "Screaming fast Galois field arithmetic using Intel SIMD instructions," in *FAST*, 2013, pp. 299–306.

[3] T. Schwarz, D. D. Long, and J. F. Pâris, "Reliability of disk arrays with double parity," in *Dependable Computing (PRDC), 2013 IEEE 19th Pacific Rim International Symposium on*. IEEE, 2013, pp. 108–117.

[4] L. Xu, V. Bohossian, J. Bruck, and D. G. Wagner, "Low-density MDS codes and factors of complete graphs," *Information Theory, IEEE Transactions on*, vol. 45, no. 6, pp. 1817–1826, 1999.

[5] J. Zhou, G. Wang, X. Liu, and J. Liu, "The study of graph decompositions and placement of parity and data to tolerate two failures in disk arrays: Conditions and existance," *Chinese Journal of Computers (chinese edition)*, vol. 26, no. 10, pp. 1379–1386, 2003.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 391 of 411

ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/221397078

Duplicate Data Elimination in a SAN File System.

Conference Paper · January 2004

Source: DBLP

CITATIONS
54

READS
243

4 authors, including:



Demyn Plantenberg
IBM
5 PUBLICATIONS **70** CITATIONS
SEE PROFILE



Darrell D. E. Long
University of California, Santa Cruz
307 PUBLICATIONS **8,768** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Fault-tolerant archival storage arrays View project

REINAS View project

All content following this page was uploaded by Darrell D. E. Long on 26 May 2014.

The user has requested enhancement of the downloaded file.

Darell Long, Ph.D.
09/17/2025
Exhibit L
H. Bautista, CSR #11600

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 392 of 411

Duplicate Data Elimination in a SAN File System

Bo Hong
Univ. of California, Santa Cruz
hongbo@cs.ucsc.edu

Demyn Plantenberg
IBM Almaden Research Center
demyn@almaden.ibm.com

Darrell D.E. Long
Univ. of California, Santa Cruz
darrell@cs.ucsc.edu

Miriam Sivan-Zimet
IBM Almaden Research Center
mzimet@us.ibm.com

Abstract

Duplicate Data Elimination (DDE) is our method for identifying and coalescing identical data blocks in Storage Tank, a SAN file system. On-line file systems pose a unique set of performance and implementation challenges for this feature. Existing techniques, which are used to improve both storage and network utilization, do not satisfy these constraints. Our design employs a combination of content hashing, copy-on-write, and lazy updates to achieve its functional and performance goals. DDE executes primarily as a background process. The design also builds on Storage Tank's FlashCopy function to ease implementation.[1]

We include an analysis of selected real-world data sets that is aimed at demonstrating the space-saving potential of coalescing duplicate data. Our results show that DDE can reduce storage consumption by up to 80% in some application environments. The analysis explores several additional features, such as the impact of varying file block size and the contribution of whole file duplication to the net savings.

1 Introduction

Duplicate data can occupy a substantial portion of a storage system. Often the duplication is intentional: files are copied for safe keeping or for historical records. Just as often, the duplicate data appear through independent channels: individuals who save the same email attachments or who download the same files from the web. It seems intuitive that addressing all of this unrecognized redundancy could result in storage resources being used more efficiently.

Our research goal is to reduce the amount of duplicated data in on-line file systems without significantly impacting system performance. This performance requirement is what differentiates our approach, which we call Duplicate Data Elimination (DDE), from those used in backup and archival storage systems [1, 7, 23]. To minimize its performance impact, DDE executes primarily as a background process that operates in a lazy, best-effort fashion whenever possible. Data is written to the file system as usual, and then some time later, background threads find duplicates and coalesce them to save storage. DDE is transparent to users. It is also flexible enough to be enabled and disabled on an existing file system without disrupting its operation, and flexible enough to be used on parts of the file system, such as select directories or particular file types.

Duplicate Data Elimination (DDE) is designed for IBM Storage Tank [17], a heterogeneous, scalable SAN file system. In Storage Tank, file system clients coordinate their actions through meta-data servers, but access the storage devices directly without involving servers in the data path. DDE uses three key techniques to address its design goals: content-based hashing, copy-on-write (COW), and lazy update. DDE detects duplicate data on the logical block level by comparing hashes of the block contents; it guarantees consistency between block contents and their hashes by using copy-on-write. Data is coalesced by changing corresponding file block allocation maps. COW and lazy update allow us to update the file block allocation maps without revoking the file's data locks. Together these techniques minimize DDE's performance impact.

Figure 1 shows an example of coalescing duplicate data blocks in an on-line file system. Before coalescing, files F_1, F_2 and F_3 consume 11 blocks. However, they each contain a common piece of data that is three blocks in size. Clients are unaware of this duplication when they write these files. The server detects the common data later and coalesces the identical blocks. After coalescing, F_1–F_3 consume only five blocks in total by sharing the same three blocks. Six blocks are saved, resulting in a 55% storage reduction.

[1] Storage Tank technology is available today in the IBM Total Storage SAN File System (SANFS). However, this paper and research is based on underlying Storage Tank technology and may not become part of the IBM TotalStorage SAN File System product.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 393 of 411

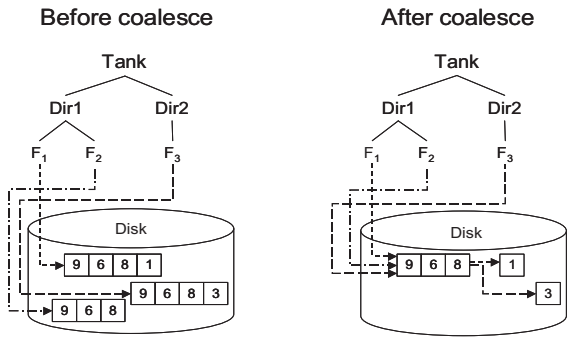


Figure 1. An example of coalescing duplicate data blocks.

2 Background

IBM Storage Tank is a multi-platform, scalable file system that works with storage area networks (SANs) [17]. In Storage Tank, data is stored on devices that can be directly accessed through a SAN, while meta-data is managed separately by one or more specialized Storage Tank meta-data servers. Storage Tank clients are designed to direct all meta-data operations to Storage Tank servers and to direct all data operations to storage devices. Storage Tank servers are not involved in the data path.

The current version of Storage Tank works with ordinary block-addressable storage devices such as disk drives and RAID systems. The basic I/O operation unit in Storage Tank is a block. The storage devices are required to have no more intelligence than the ability to read and write blocks from the volumes (LUNs) they present. Storage Tank file data is also managed in block units. The size of a file block is typically a multiple of the device block size.

Storage Tank exposes three new abstractions called *file sets*, *storage pools*, and *arenas*. These are in addition to the traditional abstractions found in file systems such as files, directories, and volumes. A file set is a subtree of the global namespace. It groups a set of Storage Tank files and directories for the purpose of load balancing and management. A storage pool is a collection of one or more volumes. It provides a logical grouping of the volumes for the allocation of space to file sets. A file set can cross multiple storage pools. An arena provides the mapping between a file set and a storage pool. As such, there is one arena for each file set that has files in a particular storage pool. The arena abstraction is strictly internal to the Storage Tank server, but is an important element in duplicate data elimination. Using an arena, Storage Tank can track the used and free space owned by a file set in a storage pool, and specifies the logical to physical mapping of space in the file set to the volumes in the storage pool.

The Storage Tank Protocol provides a rich locking scheme that enables file sharing among Storage Tank clients or, when necessary, allows clients to have exclusive accesses to files [6, 8, 9]. A Storage Tank server grants locks to clients, and the lock granularity is per file. There are three file lock modes in Storage Tank: 1) exclusive (X), which allows a single client to cache both data and metadata, which it can read and modify; 2) Shared Read (SR), which allows clients to cache data and metadata for read operations, and 3) Shared Write (SW), in which mode clients cannot cache data but can cache metadata in read-only mode. The Storage Tank Protocol also provides copy-on-write capability to support file system snapshots. The server can mark blocks as read-only to enforce copy-on-write.

Storage Tank technology is available today in the IBM Total Storage SAN File System (SANFS). However, this paper and research is based on underlying Storage Tank technology and may not become part of the IBM TotalStorage SAN File System product.

3 Related Work

Data duplication is ubiquitous. Different techniques have been proposed to identify commonality in data, and to exploit this knowledge for reducing storage and network resource consumption due to storing and transferring duplicate data.

Our work was directly inspired by Venti [23]. Venti is a network storage system intended for archival data. In Venti, the unique SHA-1 [3] hash of a block acts as the block identifier, which is used in place of the block address for read and write operations. Venti also implements a write-once policy that prohibits data from being deleted once it is stored. This write-once policy becomes practical, in part, because Venti stores only one copy of each unique data block.

In on-line file systems, performance is essential and data is dynamic and ready to change. This is radically different from the requirements in archival and backup systems, where data is immutable and performance is less of a concern. In our design, duplicate data is also detected on the block level, but in the background. Data is still addressed as usual by the block where it is stored, so data accesses have no extra hash-to-block index searching overheads as in Venti. In turn, it determines that data duplication detection and coalescing is an after-effect effort, *i.e.* it is done after clients have written data blocks to storage devices. The server also maintains a mapping function between block hashes and blocks. A weaker variant of the write-once policy, copy-on-write (COW), is used to guarantee its consistency. Unreferenced blocks due to deletion and COW can be reclaimed.

Single instance store (SIS) [4] also detects duplicate data in an after-effect fashion but on the file level. The technique is optimized for Microsoft Windows remote install servers [18] that store different installation images. In this scenario, the knowledge of file duplication is a priori and files are less likely to be modified. In general on-line file

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 394 of 411

systems, the granularity of file-level data duplication detection may be too coarse because any modification to a file can cause the loss of the benefit of storage reduction.

LBFS [20] and Pastiche [10] detect data duplication at the granularity of variable-sized chunks, whose boundary regions, called *anchors*, are identified by using the techniques of shingling [16] and Rabin fingerprints [24]. This technique is suitable for backup systems and low-bandwidth network environments, where the reduction of storage and network transmission is more important than performance.

Delta compression is another technique that can effectively reduce duplicate data, thus the requirements of storage and network bandwidth [1, 7, 19, 25]. When a base version of a data object exists subsequent versions can be represented by changes (deltas) to save both storage and network transmission. Delta compression, in general, requires some prior knowledge of data object versioning. It cannot explore common data across multiple files and a change of a (base) file may cause recalculating deltas for other files. In DERD (Delta-Encoding via Resemblance Detection) [12], similar files can be identified as pairs by using data similarity detection techniques [5, 16] without having any specific prior knowledge.

Some file systems provide on-line compression capability [15, 21]. Although it can effectively improve storage efficiency, this technique has significant run-time compression and decompression overheads. On-line compression explores intra-file compressibility and cannot take advantage of common data across files.

The techniques of naming and indexing data objects based on their content hashes are also found in several other systems. In the SFS read-only file system [13], blocks are identified by their SHA-1 hashes and the block hashes are hashed recursively to build up more complex structures. The Stanford digital library repository [11] uses the cyclic redundancy check (CRC) values of data objects as their unique handles. Content-derived names [2, 14] take a similar approach to address the issue of naming and managing reusable software components.

4 Design of Duplicate Data Elimination

Our design goal is to transparently reduce duplicate data in Storage Tank as much as possible without penalizing system performance significantly. Instead of finding data duplication at the first spot, we delay this duplication detection and identify and eliminate duplicate data when server loads are low. In this way, we minimize the performance impact of duplicate data elimination (DDE). In our design, we use three techniques: content-based hashing, copy-on-write (COW), and lazy update.

Duplicate data blocks are detected by the Storage Tank server. A client uses a collision-resistant hash function to digest the block contents it writes to storage devices and returns their hashes to the server. Such a unique hash is called the *fingerprint* of a block. The server compares block fingerprints and coalesces blocks with the same fingerprint (and the same content) by changing corresponding file block allocation maps. The server guarantees consistency between block contents and their fingerprints by directing clients to perform copy-on-write. The server also maintains a reference count for each block and postpones the reclamation of unreferenced blocks. These techniques allow the server to update file block allocation maps without revoking any outstanding data locks on them.

Figure 2 shows the basic idea of duplicate data block elimination in a live file system. The client holds an exclusive (X) lock on file A and a shared-read (SR) lock on file B. These lock modes are described in Section 2. File A and B have the same data stored in blocks 100 and 150 respectively. Before duplicate block coalescing, the server and the client share the same view of files and file block allocation maps. The server finds duplicate data and changes the block allocation map of file B to reference block 100 without updating the client. Even though the client has a stale view on file B, it can still read out the correct data because block 150 is not reclaimed immediately. When the client modifies block 100 of file A, it writes the new content to another block and keeps the content of block 100 intact. Therefore, the content and the fingerprint of block 100 are still consistent and file B still references the right block.

4.1 Duplicate Data Detection

The most straightforward and trusted way of duplicate data detection is bytewise comparison. Unfortunately, it is expensive. An alternative method is to digest data contents by hashing them to much shorter fingerprints and detect duplicate data by comparing their fingerprints. As long as the probability of hash collisions is vanishingly small, we can be confident that two sets of data content are identical if their fingerprints are the same.

In our design, duplicate data is detected on the block level, although maintaining a fingerprint for each block imposes a large amount of bookkeeping information on the system. Storage Tank is based on block-level storage devices, and blocks are the basic operation units. Block-level detection avoids unnecessary I/Os required by other approaches based on files [4] or variable-sized chunks [10, 20], in which the client may have to read other disk blocks before it can recalculate fingerprints due to data boundary mis-alignments. Data duplication detection that is based on blocks has finer granularity and, therefore, higher possibility of storage reduction than techniques based on whole files [4]. Approaches based on variable-sized chunks [10, 20] require at least as much bookkeeping information as the block-based approaches because they tend to limit the average chunk size to obtain a reasonable chance of detecting duplicate data. Chunk-based approaches also need a totally different file block allocation map format from the existing

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 395 of 411

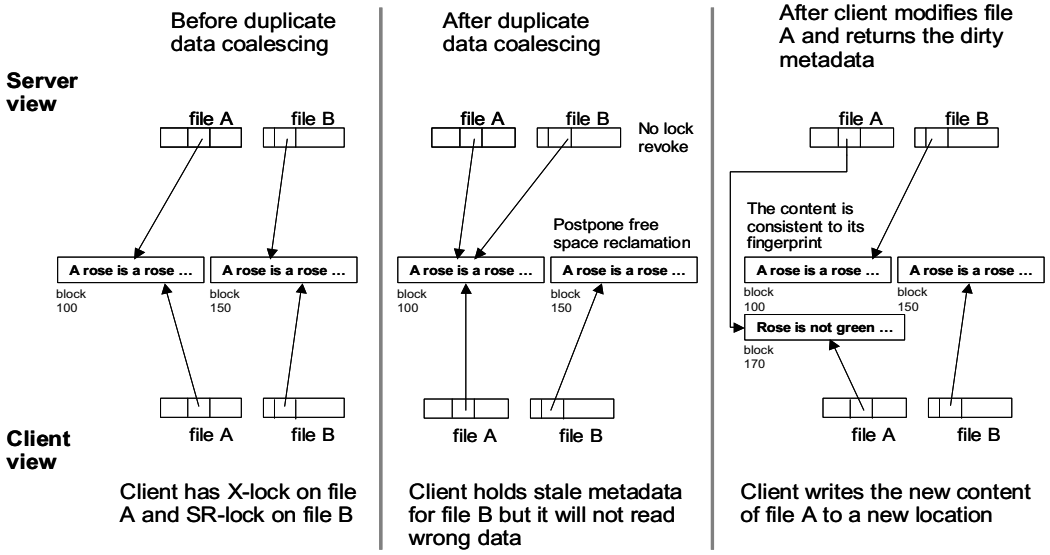


Figure 2. An example of coalescing duplicate data blocks in a live file system.

Storage Tank implementation, which makes them difficult to employ.

DDE uses the SHA-1 hash function [3] to fingerprint block data contents. SHA-1 is a one-way secure hash function with a 160-bit output. Even in a large system that contains an exabyte of data (10^{18} bytes) as 4 kilobyte blocks (roughly 3×10^{14} blocks), the probability of hash collisions using the SHA-1 hash function is less than 10^{-19}, which is at least 5–6 orders of magnitude lower than the probability of an undetectable disk error. To date, there are no known collisions by this hash function. Therefore we can be confident that two blocks are identical if their SHA-1 hashes are the same. In addition, the system could perform bytewise comparisons before coalescing blocks as a cross check.

In Storage Tank, data and meta-data management are separated, and Storage Tank servers are not involved in the data path during normal operations. Disks have little intelligence and cannot detect duplicate data by themselves. Even with smarter disks, without extensive inter-disk communications, each disk would know only its local data fingerprints, which would reduce the chances of detecting duplication. In our design, a client calculates fingerprints of the blocks it writes to storage devices and returns them to the server. Software implementations of SHA-1 are quite efficient and hashing is not a performance bottleneck. Storage Tank servers have a global view of the whole system and are appropriate for data duplication detection.

4.2 Consistency of Data Content and Fingerprints

After we hash the data content of a block, the fingerprint becomes an attribute of the block. Because the fingerprint is eventually stored on the server and the block can be directly accessed and modified by clients, the consistency of the fingerprint and the data content of a block becomes a problem. In Storage Tank, a client can modify a block by two approaches: update-in-place and copy-on-write.

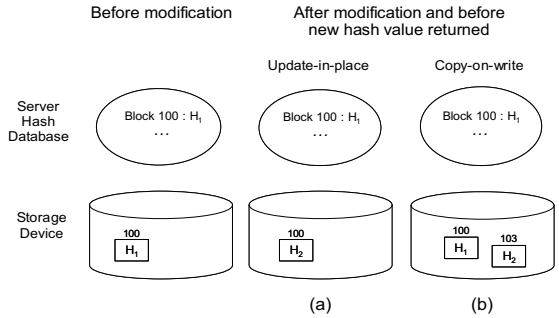


Figure 3. Maintaining consistency between fingerprint and block content under (a) update-in-place and (b) copy-on-write.

4.2.1 Update-in-Place

In Storage Tank, a client can directly modify a block if the block is writable, *i.e.* it writes new data to the same block. This results in inconsistency between the server-side block fingerprint and the block content until the client returns the new fingerprint to the server, as shown in Figure 3(a). The fingerprint of block 100 that the server keeps is inconsistent with the block content until the client returns the latest fingerprint. During the period of inconsistency, any data duplication detection related to this block gives false results.

We can detect this potential inconsistency by checking data locks on the file to which the block belongs. Because the granularity of file data locks in Storage Tank is per file, the server cannot trust all of the block fingerprints of a file with an exclusive data lock on it.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 396 of 411

To avoid erroneous duplication detection and coalescing, we could simply delay DDE on those files with exclusive or shared-write locks. This is feasible in some workloads where only a small fraction of files are active concurrently. However, this approach cannot save any storage in some important environments, such as databases, where applications always hold locks on their files.

Another approach is to revoke data locks on files to force clients to return the latest block fingerprints. This causes two technical problems: lock checking and lock revocation. To check file locks, every block has to maintain a reverse pointer to the file to which it belongs, which makes the bookkeeping information of a block even larger. To guarantee consistency between fingerprints and block contents, every block-coalescing operation has to revoke file locks, if necessary, which can severely penalize the system performance. Therefore, eliminating duplicate data blocks under the update-in-place scenario is inefficient, at best, or impossible.

4.2.2 Copy-on-Write

The basic idea of our work is to eliminate duplicate data blocks by comparing their fingerprints. By using a collision-resistant hash function with a sufficiently large output, such as SHA-1, the fingerprints are considered to be distinct for different data. Therefore, the fingerprint can serve as a unique virtual address for the data content of a block. The mapping function from the virtual address to the physical block address is implicitly provided by the block address itself. Our aim is to make the mapping function nearly one-to-one, *i.e.* each virtual address is mapped to only one physical address.

However, update-in-place violates the basic concept of content-addressed storage by making the mapping function inconsistent. Conceptually, if the content of a block is changed, the new content should be mapped to a new block instead of the original one. Consequently, a client should write modified data to new blocks, which implies a write-once policy, as in Venti [23]. However, write-once keeps all histories of data, which is unnecessary and expensive in on-line file systems. Therefore, we use a weaker variant of write-once, copy-on-write. This technique can guarantee the consistency of the mapping function as long as the original blocks are not reclaimed, as shown in Figure 3(b). The fingerprint of block 100 that the server keeps is still consistent with the block content until block 100 is reclaimed.

Apparently, copy-on-write has a noticeable overhead on normal write operations. Every block modification requires free block allocation because the modified content needs to be written to a new block. However, this cost is less significant than we thought. Some applications, such as Emacs and Microsoft Word, write the whole modified file into a new place, in which case there is no extra cost for COW. The server could also preallocate new blocks to the client that acquires an exclusive or shared-write lock. The most promising approach to alleviate the extra allocation overhead of COW is for the clients to maintain a private storage pool on behalf of the server from which they could allocate locally. Therefore, there is almost no extra cost for COW.

4.3 Lazy Lock Revocation and Free Space Reclamation

The server coalesces duplicate data blocks and reduces storage consumption by updating file block allocation maps to point to one copy of the data and reclaim the rest. During file block allocation map updates, the server does not check whether any client holds data locks on the files. Therefore, the block allocation maps held by the server and clients can be inconsistent, as illustrated in Figure 2. However, we postpone the reclamation of the dereferenced blocks. Therefore, clients holding stale file block allocation maps can still read the correct data from these blocks. At some particular time, *e.g.* midnight, or when the file system is running low on free space, the server revokes all data locks held by clients and frees those dereferenced blocks.

5 Process of Duplicate Data Elimination

Duplicate data elimination is done by the coordination between clients and servers. Simply speaking, clients perform copy-on-write operations and calculate and return block SHA-1 hashes to servers. Servers log clients' activities and identify and coalesce duplicate data blocks in the background. Users are unaware of such operations.

5.1 Impact on Client's Behaviors

In addition to its normal behaviors, a client calculates SHA-1 fingerprints for the data blocks it writes and returns the fingerprints to the server. Because copy-on-write is used (Section 4.2.2), the client does not write modified data blocks back to their original disk blocks; instead, modified data is written to newly-allocated blocks. As long as the client holds a file data lock, further modifications to the same logical block are written to the same newly-allocated disk block. On an update to the server, the client sends the latest block fingerprints along with the block logical offsets within the file and the original physical locations of modified data blocks.

5.2 Data Structures on the Server

We discuss necessary data structure supports on the server that facilitate duplicate data block detection and elimination. Essentially, a reference count table, a fingerprint table and its secondary index maintain attributes associated with blocks, *i.e.* reference counts and fingerprints, as shown

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 397 of 411

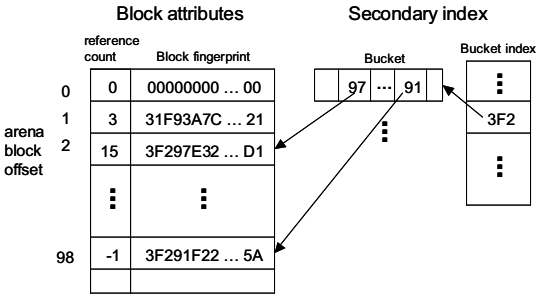


Figure 4. Data structures for storing and retrieving block attributes.

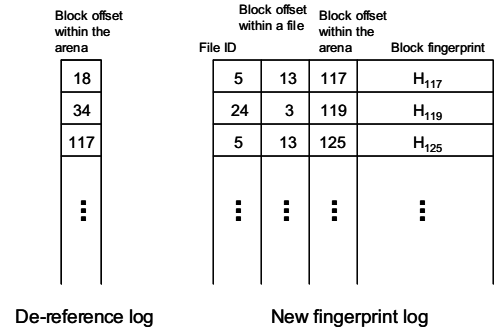


Figure 5. Data structures for logging recent clients' activities.

in Figure 4; and a dereference log and a new fingerprint log record recent clients' activities, as shown in Figure 5.

The scope of duplicate data elimination is an important design decision. The larger the scope, the higher the degree of data duplication can be, thus providing more benefit. However, for various reasons, people may want to share data only within their working group or their department. Therefore, we limit data duplication detection and elimination within a file set, which essentially is a logical subset of the global namespace. Even within a file set, files can be stored in different storage pools that may belong to different storage classes, which have different characteristics in access latency, reliability, and availability. Sharing data across storage classes can result in noticeable impacts on the quality of storage service. Therefore, we further narrow the scope of DDE within an arena, which provides the mapping between a file set and a storage pool. The data structures we will discuss soon are per arena.

Data are not equally important. Detecting and coalescing temporary, derived, or cached data is less beneficial. Because Storage Tank provides policy-based storage management, a system can be easily configured to store these data in less reliable and so cheaper storage pools, while storing important data in more reliable storage pools. DDE within an arena can take advantage of this flexibility.

Because an arena allows the logical to physical mapping of space in the file set to the LUNs in the storage pool, it

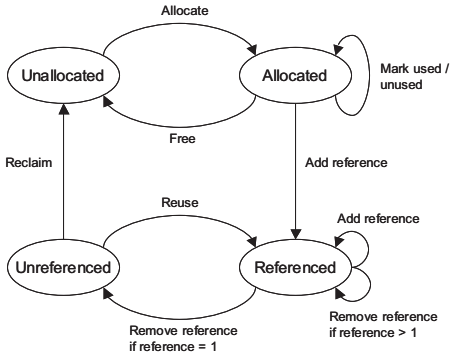


Figure 6. A block is in one of the following states: unallocated, allocated, referenced, and unreferenced.

is equivalent to referencing a block by its physical location and by its logical offset within an arena. For convenience, we will reference an allocated physical block by its logical offset within the arena.

To keep the per-arena data structures to an optimal size, we use 32-bit integers to represent logical block offsets within an arena. Therefore, an arena can contain no more than 2^{32} physical blocks. For 4 KB blocks, an arena can manage 16 TB storage, which is large enough for most applications and environments. However, there is no such a limitation on the capacity of a file set because it can cross multiple storage pools and can consist of multiple arenas.

5.2.1 Reference Count Table

With block coalescing and sharing, a physical block can be referenced multiple times by different files or even one file. Therefore, a reference count is necessary for each block in the arena. From the viewpoint of DDE, a block can be in one of the following four states: 1) free – the block is unallocated; 2) allocated – the block is allocated but unused or it contains valid data that is unhashed; 3) referenced – the block contains valid data, which has been hashed, and is referenced at least once; and 4) unreferenced – the block is allocated and hashed, but has no file referencing it and can be freed or reused. Figure 6 illustrates the four states and the transitions among them.

Without accessing the arena block allocation map, we cannot know whether a block is allocated or not. Fortunately, we are interested in the validity of block fingerprints in our work. Therefore, blocks that are in the first two states (free and allocated) can be merged into one state, invalid, because they contain no valid fingerprints. The reference count of a block indicates its state: 1) invalid, where the reference count is 0, 2) referenced, where the reference count is no less than 1, or 3) unreferenced, where the reference count is -1. The initial state of a block is invalid because it contains no valid fingerprint until a client calculates it.

A reference count table keeps the reference count for each block in the arena. The table is organized as a linear

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 398 of 411

array, which is indexed by the 32-bit logical block offset within the arena. Each entry in this table is also a 32-bit integer, indicating the state of the corresponding block. The size of the table is up to $2^{32} \times 4$ bytes $= 16$ GB. Because block reference counts are crucial to data integrity, any update on them should be transactional.

A block may contain valid data but has no fingerprint associated with it. Note that the fingerprint of a block is calculated when it is written. If a block is written before the server turns on this feature, it has no fingerprint on the server. A utility running on the server could ask clients to read those data blocks and calculate their fingerprints on behalf of the server.

5.2.2 Fingerprint Table

The fingerprint table keeps unique fingerprints of the blocks in an arena. Each fingerprint in this table is associated with a unique physical block. In other words, the table maintains a one-to-one mapping function between fingerprints and physical blocks. We detect and coalesce duplicate data blocks when we merge the new fingerprint log to this table. The fingerprint table is also organized as a linear array and indexed by the 32-bit logical block offset. Each entry contains a 160-bit SHA-1 fingerprint. A fingerprint is valid only if its block reference count is no less than 1.

The size of the table is up to $2^{32} \times 20$ bytes $= 80$ GB, and it cannot fit in memory. Fortunately, disk block accesses have sequential patterns due to sequential block allocations and file accesses. Therefore, we organize the secure fingerprint table linearly to facilitate comparisons under sequential block accesses. If two disk blocks contain the same content, it is likely that their consecutive blocks also have identical contents. The linear structure makes consecutive fingerprint comparisons efficient because all related entries are in memory.

Conceptually, both the reference count table and the fingerprint table are for describing block attributes and can be merged into one table. Because their sizes are potentially large, and the block reference count is accessed more frequently than the fingerprint, we store them separately to optimize system memory usage.

5.2.3 Secondary Index to Fingerprint Table

Although the linear fingerprint table favors sequential searching, it is difficult to look for a particular fingerprint in this table. Therefore, we also index the table by partial bits of the SHA-1 fingerprint to facilitate random searching. A static hash index is used for this purpose. The hash buckets are indexed by the first 24 bits of the SHA-1 fingerprint. Each bucket contains a 32-bit block pointer. Therefore, the size of the first level index is $2^{24} \times 4$ bytes $= 64$ MB, which can well fit in memory. Each entry in the bucket block contains a 32-bit in-arena logical block offset, indicating the block that the fingerprint is associated with, and the next 32 bits of the SHA-1 fingerprint. Because an arena contains no more than 2^{32} blocks, the average number of hash entries in a bucket is $2^{32}/2^{24} = 2^8 = 256$. When the bucket block size is 4 KB, the average block utilization is $(256 \times (4+4))/4096 = 50\%$. For an arena with capacity much less than 16 TB, multiple buckets can share one bucket block for better storage and memory utilization.

5.2.4 Dereference Log

The dereference log records the in-arena logical offsets of blocks that are recently deleted, dereferenced due to COW by clients, or dereferenced due to block coalescing by the server. We will discuss the third case in greater details in Section 5.3.3. This log is also called *semi-free list* because the blocks in this list could be freed if there is no longer any reference to them. Each entry in this log is a 32-bit integer. To avoid storage leakage, any update on this log should be transactional.

5.2.5 New Fingerprint Log

The new fingerprint log records clients' recent write activities. Each entry in this log includes a 64-bit file ID, a 64-bit logical block offset within the file, a 32-bit logical offset within the arena, and a 160-bit SHA-1 fingerprint. Appending new entries to this log can be non-transactional for the performance purpose because losing the most recent entries only causes losing some opportunities for storage reduction.

5.3 The Responsibilities of the Server

The server enforces a client's copy-on-write behaviors by marking the copy of the file block allocation map in the message buffer as read-only when it responds to a file data lock request from the client. The server immediately logs clients' recent activities, such as delete and write operations. In our design, DDE runs in a best-effort fashion. Therefore, the server lazily detects and coalesces duplicate data blocks and reclaims unused blocks. It also maintains block reference counts and fingerprints correspondingly.

5.3.1 Logging Recent Activities

When the server receives a dirty file block allocation map from a client, it compares it with the one on the server. If a block is marked as unused due to copy-on-write, the server first checks whether it is still referenced by the server-side file block allocation maps. If referenced, the in-arena logical offset of the unused block is appended to the dereference log; if not (it is possible due to duplicate block coalescing without lock revocation), the block currently referenced by the server-side block allocation map is logged because it is dereferenced by recent modifications on the corresponding file logical block. The unused block returned from the client is not logged because it has been dereferenced due to block coalescing. Other unused blocks are logged.

For each recently-written block, the server also appends an entry to the new fingerprint log, including the identifier of the file to which it belongs, its logical block offset within the file, its logical block offset within the arena, and its fingerprint.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 399 of 411

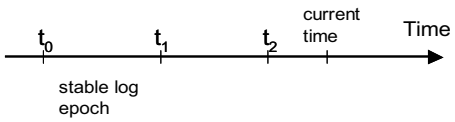


Figure 7. Log epochs.

5.3.2 Log Epoch and Preprocessing

We periodically checkpoint the dereference log and the new fingerprint log for two reasons. First of all, the data duplication detection and elimination processes run in a best-effort and background fashion and are unlikely to keep up with the most recent clients' activities. Logging and checkpointing these activities allow the server to detect and coalesce duplicate data blocks during its idle time. By checkpointing the log to epochs, we also limit the number of activities the server processes at a time. Second, recently-written blocks are likely to be modified again. Trying to coalesce these blocks is less beneficial. Therefore, we try to coalesce only blocks in a stable epoch, as shown in Figure 7, whose lifetimes have been long enough.

Assume that we want to merge the new fingerprint log in epoch (t_0, t_1) to the fingerprint table. Because random accesses on the fingerprint table are expensive, we try to reduce the number of fingerprint comparisons by deleting unuseful entries in the new fingerprint log in epoch (t_0, t_1).

First of all, we find overwritten file logical blocks by sorting the log by file ID and logical block offset within a file. We delete the older entries from the log and set their block reference counts to be -1 (unreferenced). We set the block reference counts of other entries in the log to be 1. Second, we scan the dereference log in epoch (t_0, t_1). For each entry in the log, we decrease its block reference count by 1; if the count becomes less than 1, we set it to be -1. Third, we compact the new fingerprint log (t_0, t_1) by deleting those entries also in the dereference logs (t_0, t_1) and (t_1, t_2). The matched entries in the dereference log (t_1, t_2) are removed and their block reference counts are set to be -1.

Note that a fingerprint in the fingerprint table becomes invalid when its block reference count reaches -1. Conceptually, we should also remove its index entry in the secondary index. However, we do not update the secondary index during the log preprocessing because of performance reasons. We postpone the removal of false indexes until the duplicate block coalescing process notices them. False index removal also happens when a secondary index bucket block becomes full.

5.3.3 Merging to Fingerprint Table

We detect and coalesce duplicate data blocks when we merge the compacted new fingerprint log to the fingerprint table. Figure 8 shows the processes of duplication detection and coalescing.

For each entry in the log, we first check whether it has a matching fingerprint in the fingerprint table. If not, we insert the new fingerprint into the table and update the secondary index. If there is an identical fingerprint in the fingerprint table, we check the validity of the fingerprint. If the block reference count of the fingerprint is less than 1, the primary block in the fingerprint table contains no valid data. Therefore, we insert the new fingerprint into the table and update the secondary index. We also need to delete the false index in the secondary index because the previous matching index leads to a block containing invalid data. If the block reference count is no less than 1, we fetch the block allocation map of the file to which the recently-written block belongs, and check whether this block is still referenced by this file. If not, this means that the corresponding logical block in this file was modified after the current fingerprint was returned, and we simply discard this coalescing operation. If the block is still referenced, we update the file block allocation map by referencing the primary block in the fingerprint table without checking or revoking data locks on this file. We also increase the reference count of the primary block and set the reference count of the coalesced block to be -1. When a block is inserted into the fingerprint table, either by adding a new entry or by coalescing to another block, it is marked read-only in the block allocation map of the file to which it belongs.

5.3.4 Free Space Reclamation

A free space reclamation process scans the reference count table in the background. It logs the addresses of the blocks with reference counts -1 and sets their reference counts to 0. At some particular time, *e.g.* midnight, or when the file system is running low on free space, it revokes all data locks and free these blocks.

6 Case Studies

We examined six data sets and studied their degrees of data duplication and their compressibility under common compression techniques. The data sets are summarized in Table 1.

The first three data sets—SNJGSA, BVRGSA_BUILD, and BVRGSA_TEST—are from file servers used by the Storage Tank development team. The servers are used to distribute and exchange data files, but are not used to hold or archive the primary copy of important files. The first server, SNJGSA, is a remote replica that holds a subset of the daily builds that are stored on BVRGSA_BUILD, and a subset of the test data that is on BVRGSA_TEST. In general, the oldest files are deleted when the servers run low on space. The files are almost never overwritten; they tend to be created and then not modified until they are purged few months later.

GENOME contains the human being genome sequence, and is used at UCSC by various bioinformatics applications. The genomic data is encoded in letters, when some single letters and some letter combinations can repeat thousands to millions of times but in fine granularities.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 400 of 411

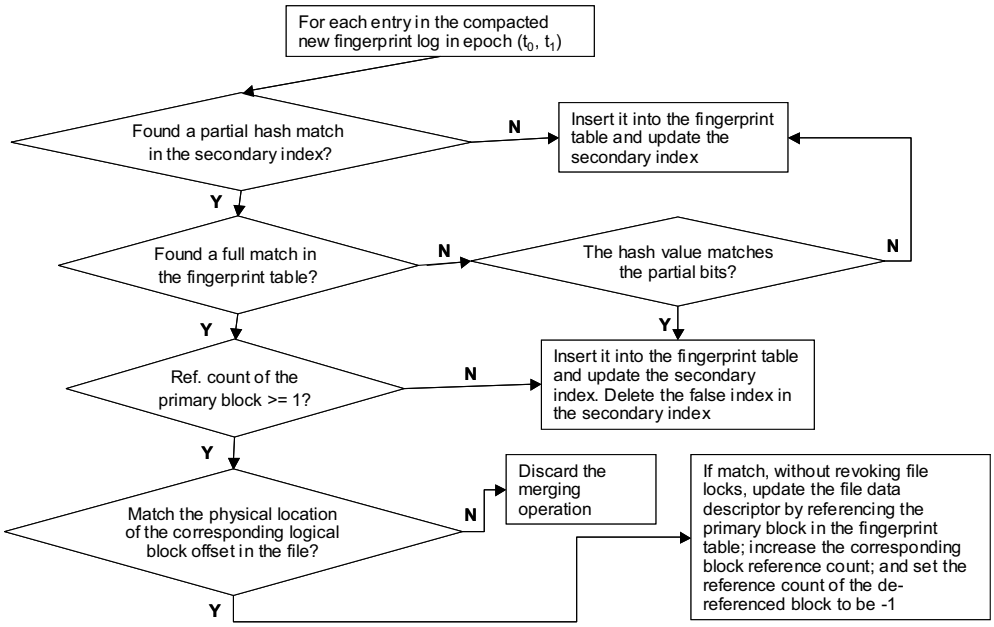


Figure 8. Merge the new fingerprint log to the fingerprint table.

Table 1. Data sets

Name	Description	Size (GB)	Number of files
SNJGSA	file server used by a development team	57	661,729
BVRGSA_BUILD	file server used by the development team for code build	344	2,393,795
BVRGSA_TEST	file server used by the development team for testing	215	115,141
GENOME	human being genome data	348	889,884
LTC_MIRROR	local mirror of installation CDs for different Linux versions	261	241,724
PERSONAL_WORKSTATIONS	aggregation of ten personal workstations	123	879,657

LTC_MIRROR is a local ftp mirror of the IBM Linux Technology Center (LTC) that includes Open Source and IBM software for internal download and use. Among other things, the ftp site holds the CD images (iso) of different Red Hat Linux installations starting at RH7.1 up to RH9.

The last data set is an aggregation of ten personal workstations at the IBM Almaden Research Center that are running Windows. All systems are used for development as well as for general purposes such as email, working documents, etc. We also present the results of these systems when they are analyzed separately.

For each data set, we collected the size and the number of files of the system. We calculated the amount of storage that is required after eliminating data duplication at the granularity of 1 KB blocks. To compare DDE with common compression techniques, we collected the compressed file sizes, using LZO on 64 KB data blocks. LZO (Lempel-Ziv-Oberhumer) is a data compression library that favors speed over compression ratio [22]. We empirically found that the compression capability of LZO is similar to other techniques' when the block size is large. In addition, we calculated the storage reduction achieved by combining the techniques of DDE and LZO. We also calculated what percentage of the storage is still required after eliminating only whole file duplications.

Table 2 shows the percentage of storage required for different data sets after using DDE at the granularities of 1 KB blocks and whole files, LZO on 64 KB blocks, and the combination of DDE and LZO. DDE at 1 KB blocks only requires one-fifth to one-third of the original storage to hold the BVRGSA_BUILD and SNJGSA data sets and it achieves one to two times higher storage efficiency (the ratio of the amount of logical data to the amount of required physical storage) than LZO at 64 KB blocks. This is because both data sets contain daily builds of the Storage Tank codes that share lots of codes among versions; also the data sets include very small files on average, making LZO inefficient. BVRGSA_TEST on the other hand, has on average larger files (1.9 MB) and the best reduction is achieved when using the combination of LZO compression and DDE

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 401 of 411

Table 2. DDE and compression results.

Name	% of storage required after DDE (1 KB blocks)	% of storage required after eliminating whole file duplications	% of storage required after LZO on 64 KB blocks	% of storage required after combining DDE and LZO on 64 KB blocks
SNJGSA	32%	55%	56%	30%
BVRGSA_BUILD	21%	62%	67%	35%
BVRGSA_TEST	69%	85%	53%	47%
GENOME	96%	98%	46%	44%
LTC_MIRROR	80%	94%	98%	89%
PERSONAL_WORKSTATIONS	54%	69%	63%	43%

Table 3. DDE and compression results for personal workstations.

System	% of storage required after DDE (1 KB blocks)	% of storage required after eliminating whole file duplications	% of storage required after LZO on 64 KB blocks	% of storage required after combining DDE and LZO on 64 KB blocks
1	66%	71%	61%	43%
2	61%	78%	57%	43%
3	63%	77%	55%	41%
4	77%	91%	63%	55%
5	67%	92%	62%	53%
6	69%	77%	58%	48%
7	70%	80%	61%	49%
8	71%	78%	71%	55%
9	87%	91%	80%	74%
10	73%	84%	57%	48%

on 64 KB blocks. We will study the data set of SNJGSA in greater detail in Section 6.1.

The genomic data set is encoded in letters. Some single letters and letter combinations can repeat thousands to millions of times but at quite fine granularities. With this type of data set it is unlikely to find duplications at the granularities of kilobytes and DDE cannot improve storage efficiency significantly. On the other hand, common compression techniques, such as LZO, are suitable for this data set. The 4% of reduction by DDE is partially due to common file headers and a common "special letter pattern" that fills in gaps in the genomic sequence and partially due to duplicated files that were created by analyzing applications.

Using LZO on the data set of LTC_MIRROR barely reduces storage consumption because generally the files in the Linux installation CD images have already been packaged and compressed. DDE can take advantage of cross-file duplications, mainly from different installation versions, and reduce storage consumption by 20%. A more interesting data set is the actual installation of different Linux versions, instead of the installation images. We plan to look into that in the nearest future.

We also studied the storage requirements of ten personal workstations after using different techniques on individual systems, as shown in Table 3. On average 70% of storage is required for individual systems after applying DDE on them, from which more than half of the saving is due to eliminating whole file duplications. LZO compression can provide better storage efficiency on these systems. By combining both LZO and DDE on 64 KB blocks, the percentage of storage required for individual systems is further reduced to 51% on average. When aggregating files from all machines together, which is potentially what happens with enterprise-level backup applications, the aggregated storage requirement of ten personal workstations by using DDE drops significantly, from 70% to 54%, as shown as PERSONAL_WORKSTATIONS in Table 2. This is because the more individual systems are aggregated, the higher the degree of data duplication is likely to be. Consequently, DDE can result in tremendous storage savings for back up systems that potentially backup hundreds to thousands of personal workstations on a daily basis, which shows a very good example of an application that could benefit from data duplication elimination at the file system. Combining DDE and LZO on 64 KB blocks can further reduce the storage requirement to 43%.

The granularity of duplicate data detection affects the effectiveness of DDE. Table 2 shows that file-level detection can lose up to 50–70% of the chance of finding duplicate data at 1 KB blocks. We also noticed that using both DDE and LZO on 64 KB blocks does not always generate better results than using DDE solely. In general, smaller block sizes favor DDE because of finer granularities on duplication elimination; larger block sizes favor LZO because of more efficient encoding. In fact, DDE and LZO-like compression techniques are orthogonal when they are operated at the same data unit granularity because compression explores intra-unit compressibility and DDE explores inter-unit duplications.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 402 of 411

6.1 Detailed Study on SNJGSA

Figure 9 shows the results of applying DDE to the SNJGSA data set using a range of file system block sizes. The results are given as a percentage of the data blocks that are unique. The block size varies from 512 bytes to 64 KB. As expected, the smallest block size works best because it enables the detection of shorter segments of duplicate data. Interestingly, DDE's effectiveness starts to improve again when the block size reaches 32 KB. The reason is that the file system is "wasting" space using these larger blocks, and DDE is proportionally coalescing more of this wasted space. This effect begins to outpace the reduced number of duplicate blocks due to coarser block granularities. Note that Storage Tank does not support blocks sizes smaller than 4 KB; these results are included to show the savings we are missing out on. The additional space saving potential of smaller blocks is modest: 5%, for instance, between 1 KB and 4 KB blocks in the SNJGSA1 data set.

SNJGSA's usage pattern and its "FIFO" style space management allow us to use this data to simulate a growing file system. Figure 10 shows the amount of space saved by DDE as the file system, which starts empty, grows to eventually contain all the files in the SNJGSA data set. The files are added in order of their modified times (mtime). 4 KB blocks are used. The rightmost value on the chart is 38%, which matches the space savings shown in Figure 9.

If Figure 10 had shown a gradual improvement starting at 100% (no compression) and monotonically decreasing to 38%, we could assert that duplication in this data set is independent of time. This is the type of curve that would be generated if the files were inserted in random order rather than being sorted by mtime. If the chart had shown a flat curve, this would indicate that data duplication is highly correlated in time, *i.e.* has strong temporal locality. The duplicate blocks found in a new file would likely match blocks that were added, for instance, within the last 24 hours, but very unlikely to match blocks that were added a month ago. DDE performs most efficiently on highly time correlated duplication because duplicate blocks tend to be coalesced within a log epoch and require fewer updates to the fingerprint table. At the same time, uncorrelated duplication allows DDE to produce a continually improving compression ratio as the data set grows.

The SNJGSA data set consists of 15 million 4 KB blocks, among which 3.3 million are referenced only once. Among the remaining 11.7 million blocks, only 2.5 million are unique. Figure 11 shows the cumulative distributions of storage savings made by frequently occurring data blocks. It reveals that only 1% of unique blocks contributes to 29% of the total storage savings. This suggests us that even a small amount of hash cache on the client side could explore the frequency of data duplication and save actual I/O on the first spot. Although we have not had the opportunity to study the recency of data duplication, we believe that

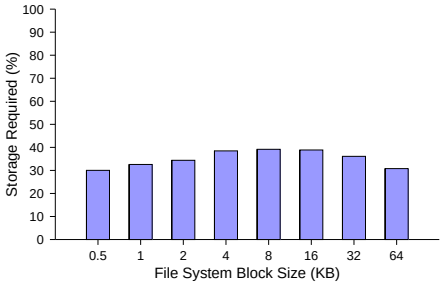


Figure 9. SNJGSA, percentage of storage required after removing duplicate data.

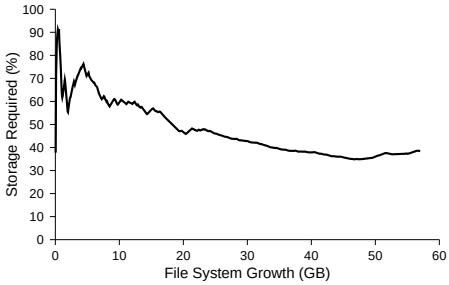


Figure 10. SNJGSA, percentage of unique blocks in a simulated growing file system.

with careful design, the client-side hash cache can also take advantage of this recency to improve system performance. Furthermore, the spatial and temporal localities of duplicate data generation are dominant factor of DDE's performance on the server side.

SNJGSA and BVRGSA_BUILD demonstrate applications that can substantially benefit from DDE. The engineers using these file servers exploit storage space to make their jobs easier. One way they do this is by creating hundreds of "views" of their data in isolated directory trees, each with a dedicated purpose. However, these pragmatic, technology savvy users do not use a snap-shot facility to create these views, or a configuration management package, or even symbolic links. The engineers employ the most robust and familiar mechanisms available to them: copying the data, applying small alterations, and leave the results to linger indefinitely. In this environment, the file system is the data management application, and DDE extends its reach.

7 Discussions

Often duplicate data copies are made for the purpose of reliability, preservation, and performance. Typically, such duplicate copies are and should be stored in different storage pools. Since the technique of duplicate data elimination is scoped to an arena, data on different storage pools will

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 403 of 411

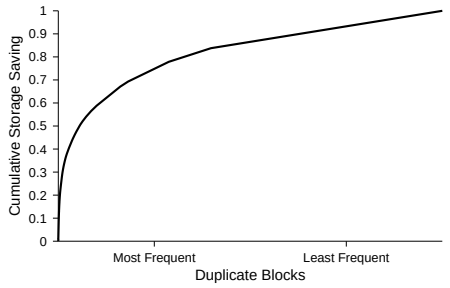


Figure 11. SNJGSA, cumulative contribution to the total storage savings made by coalescing frequently occurring data blocks. Blocks are sorted by their reference frequencies.

not be coalesced and its reliability, preservation, and the performance of accessing data will not be affected. If the duplicate data copies are made against inadvertent deletion or corruption of file data, DDE allows copies to be made without consuming additional storage space and still protects against deletion or corruption. However, DDE does create an exposure to potential multiple file corruptions due to a single block error.

DDE coalesces duplicate data blocks in files. It can result in file fragmentation thus degrade the system performance due to non-contiguous reads. This can be alleviated by only coalescing duplicate data with sizes of at least N contiguous blocks. In Section 6, we also found that the majority of duplicate blocks come from whole files, in which cases reading those files has no additional seek overheads. It is probable that DDE can reduce system write activities if clients can detect duplicate data before writing to storage devices, which will be discussed further in Section 8. DDE can also potentially improve the storage subsystem cache utilization because only unique data blocks will be cached.

The degree of data duplication can vary dramatically in different systems and application environments. DDE is a technique that is suitable for those environments with expected high degrees of data duplication such as backup of multiple personal workstations, and it is not necessarily intended for general uses.

The key techniques used in DDE are content-based hashing, copy-on-write, and lazy updates. With necessary and appropriate supports, these techniques thus DDE could be also applicable to other file systems besides Storage Tank. Particularly, lazy updates minimize the performance impact of identifying duplicate data and maintaining block metadata, which will also be beneficial to other file systems.

8 Future Work

We are working on implementing duplicate data elimination in Storage Tank. Besides implementation, there are several research directions we can explore in the future.

The technique of copy-on-write plays a key role in our design to guarantee consistency between fingerprints and block contents. However, it forces a client to request new block allocations for each file modification and has noticeable overheads on normal write operations. To alleviate the extra allocation cost due to COW, the server could preallocate new blocks to a client that acquires an exclusive or shared-write lock. Therefore, the preallocation policy for COW is an interesting research topic. Another promising approach to alleviate this overhead is to allow a client to maintain a small private storage pool on behalf of the server. Therefore, there is almost no extra cost for COW.

In our current design, we employ quite a naive coalescing policy: when we find a recently-written block containing the same data as a block in the fingerprint table, we simply dereference the new block and change the corresponding file block pointer to the primary block in the fingerprint table. This policy is suboptimal in terms of efficiency. Although we have considered file and block access patterns in our design, we do not, for simplicity, explicitly elaborate policies that favor sequential fingerprint probing and matching under certain block access patterns. Therefore, further research on such policies is needed.

The naive coalescing policy we describe in this paper may also result in file fragmentation. Coalescing few blocks within a large file is less desirable. Therefore, a study on policies for minimizing file fragmentation is interesting. Furthermore, a good coalescing policy could reduce storage fragmentation by reusing the lingering unused blocks due to lazy free space reclamation.

The fingerprint of a block is a short version of its data. A client can easily keep a history of its recent write activities by maintaining a fingerprint cache. The client can do part of the duplicate data elimination work in conjunction with the server. More beneficially, actual write operations to storage can be saved if there are cache hits.

As far as we know, there are no extensive and intensive studies on the duplicate data distributions of the block level or other levels. A better understanding of data duplication in file systems can be enormously beneficial for making good duplicate data coalescing policies.

In our design, we add two attributes on physical disk blocks: the reference count and the SHA-1 fingerprint of the block content. We also provide appropriate data structures to store and retrieve these attributes. Our work makes it feasible to check data integrity in Storage Tank. A client can ensure that the data it reads is the data it writes by comparing the fingerprint on the server with the one calculated from the data it recently reads.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 404 of 411

9 Conclusions

Although disk prices drop dramatically, storage is still a precious resource in computer systems. For some data sets, reducing storage consumption caused by duplicate data can significantly improve storage usage efficiency. By using techniques of content-based hashing, copy-on-write, lazy lock revocation, and lazy free space reclamation, we can detect and coalesce duplicate data blocks in on-line file systems without a significant impact on system performance. Our case studies show that 20–79% of storage can be saved by the technique of duplicate data elimination at 1 KB blocks in some application environments. File-level duplication detection is sensitive to changes of file contents and can lose up to 50–70% of the chance of finding duplicate data at finer granularities.

Acknowledgments

We are grateful to Robert Rees, Wayne Hineman, and David Pease for their discussions and insights in Storage Tank. We thank Scott Brandt, Ethan Miller, Feng Wang, and Lan Xue of the University of California at Santa Cruz for their helpful comments on our work. We thank Vijay Sundaram of the University of Massachusetts at Amherst, in particular, for his invaluable discussions. We also thank Terrence Furey and Patrick Gavin of the bioinformatics department of UCSC for providing access to the GENOME data set and helping us to understand the results. Our shepherd Curtis Anderson provided useful feedback and comments on our first draft.

References

[1] M. Ajtai, R. Burns, R. Fagin, D. D. E. Long, and L. Stockmeyer. Compactly encoding unstructured inputs with differential compression. *Journal of the Association for Computing Machinery*, 49(3):318–367, May 2002.

[2] K. Akala, E. Miller, and J. Hollingsworth. Using content-derived names for package management in Tcl. In *Proceedings of the 6th Annual Tcl/Tk Conference*, pages 171–179, San Diego, CA, Sept. 1998. USENIX.

[3] Anonymous. Secure hash standard. FIPS 180-1, National Institute of Standards and Technology, Apr. 1995.

[4] W. J. Bolosky, S. Corbin, D. Goebel, and J. R. Douceur. Single instance storage in Windows 2000. In *Proceedings of the 4th USENIX Windows Systems Symposium*, pages 13–24. USENIX, Aug. 2000.

[5] A. Z. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences (SEQUENCES '97)*, pages 21–29. IEEE Computer Society, 1998.

[6] R. Burns. *Data Management in a Distributed File System for Storage Area Networks*. Ph.d. dissertation, Department of Computer Science, University of California, Santa Cruz, Mar. 2000.

[7] R. Burns and D. D. E. Long. Efficient distributed back-up with delta compression. In *Proceedings of I/O in Parallel and Distributed Systems (IOPADS '97)*, pages 27–36, San Jose, Nov. 1997. ACM.

[8] R. Burns, R. Rees, and D. D. E. Long. Safe caching in a distributed file system for network attached storage. In *Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS 2000)*. IEEE, May 2000.

[9] R. Burns, R. Rees, L. J. Stockmeyer, and D. D. E. Long. Scalable session locking for a distributed file system. *Cluster Computing Journal*, 4(4), 2001.

[10] L. P. Cox, C. D. Murray, and B. D. Noble. Pastiche: Making backup cheap and easy. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 285–298, Boston, MA, Dec. 2002.

[11] A. Crespo and H. Garcia-Molina. Archival storage for digital libraries. In *Proceedings of the Third ACM International Conference on Digital Libraries (DL '98)*, pages 69–78, Pittsburgh, Pennsylvania, June 1998. ACM.

[12] F. Douglis and A. Iyengar. Application-specific delta-encoding via resemblance detection. In *Proceedings of the 2003 USENIX Annual Technical Conference*, pages 113–126. USENIX, June 2003.

[13] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 181–196, San Diego, CA, Oct. 2000.

[14] J. Hollingsworth and E. Miller. Using content-derived names for configuration management. In *Proceedings of the 1997 Symposium on Software Reusability (SSR '97)*, pages 104–109, Boston, MA, May 1997. IEEE.

[15] S. Kan. ShaoLin CogoFS – high-performance and reliable stackable compression file system. http://www.shaolinmicro.com/product/cogofs/, Nov. 2002.

[16] U. Manber. Finding similar files in a large file system. Technical Report TR93-33, Department of Computer Science, The University of Arizona, Tucson, Arizona, Oct. 1993.

[17] J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. IBM Storage Tank—a heterogeneous scalable SAN file system. *IBM Systems Journal*, 42(2):250–267, 2003.

[18] Microsoft Windows 2000 Server online help file. Microsoft Corporation, Feb. 2000.

[19] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP. In *Proceedings of SIGCOMM97*, pages 181–194, 1997.

[20] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, pages 174–187, Oct. 2001.

[21] R. Nagar. *Windows NT File System Internals: A Developer's Guide*. O'Reilly and Associates, 1997.

[22] M. F. Oberhumer. oberhumer.com: LZO data compression library. http://www.oberhumer.com/opensource/lzo/, July 2002.

[23] S. Quinlan and S. Dorward. Venti: A new approach to archival storage. In *Proceedings of the 2002 Conference on File and Storage Technologies (FAST)*, pages 89–101, Monterey, California, USA, 2002. USENIX.

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 405 of 411

[24] M. O. Rabin. Fingerprinting by random polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[25] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '00)*, pages 87–95, Stockholm, Sweden, Aug. 2000. ACM Press.

View publication stats

ParTec EX2022
Microsoft v ParTec, IPR2025-00318
Page 406 of 411

Compare Results

2025-04-09 9:28:26 PM

Old File:		New File:
20241220 PAPER 1 IPR2025-00318 Microsoft Petition for IPR '442_38.pdf	versus	**Exhibit 1003 - Long Declaration_42.pdf**
5 pages (168 KB) 2025-04-09 9:27:32 PM		**4 pages (181 KB)** 2025-04-09 9:28:08 PM

Total Changes

34

Content

20 Replacements
2 Insertions
7 Deletions

Styling and Annotations

8 Styling
5 Annotations

Go to First Change (page 1)



c) Element [1.3]

[1.3] generating, using information relating to the processing of the plurality of sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes, a further distribution of the plurality of sub-tasks between the plurality of computation nodes and the plurality of booster nodes for processing thereby in a further computing iteration.

77. Lippert-Budenske discloses and teaches this element. After processing the sub-tasks with the "initial mapping," Budenske teaches "monitor[ing] the run-time values of the dynamic parameters at the end of each iteration through the underlying DDG to make a heuristically-based decision whether to continue with the current mapping or to select and instantiate a new mapping (for the next iteration)." EX1005, 395. Budenske thus discloses using values of dynamic parameters (*information relating to the processing of the sub-tasks by at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes*) to generate a "new assignment" (*further distribution*) of sub-tasks to resources "to use for the next execution iteration through the subtasks" (*in a further computing iteration*). EX1005, 391.

78. Specifically, "[a]fter each execution iteration through the set of subtasks, the values of certain dynamic parameters of the application may change, such as the number of objects detected in the current frame of a real-time image

stream being processed." EX1005, 391. The dynamic parameters are "computed by the application as it executes." EX1005, 393. The values for the dynamic parameters are thus obtained after processing the sub-tasks in an iteration: "After all subtasks have completed execution for a given iteration through the DDG, and before the next iteration begins, the latest values of these dynamic parameters will be reported to the on-line HC Kernel." EX1005, 391. At each iteration, there can be a different mapping of the DDG (parallelism) onto the accelerators. The latest values of dynamic parameters are an example of the claim's "information relating to the processing of the plurality of sub-tasks." The "most recent values of such dynamic parameters" are then used to determine whether "to reconfigure the assignment of processing resources to subtasks to reduce execution time of the next iteration," and "[i]f it is desirable, the HC Kernel will select a new assignment to use for the next execution iteration through the subtasks." EX1005, 391. In particular, Budenske teaches generating a "vector of *D* representative values" as an "approximation of the set of current actual dynamic parameters," and using that vector to identify the optimal mapping of sub-tasks from the possible mappings (tracked in a "Mapped DDG" or "MDDG Table"). EX1005, 402. Budenske thus teaches selecting a new mapping from a set of candidate mappings that best matches the values of the dynamic parameters (*the information*), and instantiating (*generating*) the further distribution according to the selected mapping for the next computing iteration.

EX1005, 402-403. Budenske also discloses an alternative wherein a fully "on-line heuristic" is used, such that the new mapping is generated in real-time between iterations based on the information. EX1005, 389-90, 398, 403.

79. A POSITA would have found it obvious to implement the heterogeneous computing system of Lippert with the dynamic sub-task mapping method of Budenske such that, between iterations, Lippert-Budenske uses values of dynamic parameters calculated during a preceding iteration to generate a new assignment of sub-tasks to the plurality of computation nodes and the plurality of boosters for processing in the next iteration. The POSITA would have been motivated to implement Lippert-Budenske in this way partly in order "to reduce execution time of the next iteration." EX1005, 391. The POSITA would have further been motivated to implement Lippert-Budenske in this way because Budenske explains that, "[b]ecause the execution time of application subtasks … is highly input-data dependent …, this matching and scheduling of application subtasks to processors *must* be performed dynamically at run time." EX1005, 390 (emphasis added). The POSITA would have reasonably expected to succeed in implementing Lippert-Budenske partly because Lippert and Budenske each provide ample implementation details for achieving the disclosed functionality, and a POSITA would have found it simple and straightforward to combine the teachings. Lippert-Budenske thus teaches and renders this element and this claimed method obvious.

[9.5] an application manager configured to receive information from daemons operating in at least the portion of the plurality of computation nodes and at least the portion of the plurality of booster nodes to update a distribution of the multiple sub-tasks between the plurality of computation nodes and the plurality of booster nodes in a further computing iteration.

107. Lippert-Budenske discloses and teaches this element. Section X.C.2. Lippert-Budenske thus discloses and teaches all elements and renders this claimed system obvious.