UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE PATENT TRIAL AND APPEAL BOARD

DOCKER INC.,
Petitioner,

v.

INTELLECTUAL VENTURES II LLC,
Patent Owner.

Case No. IPR2025-00840

U.S. Patent No. 8,332,844

Declaration of Erez Zadok in Support of Petition for *Inter Partes* Review of U.S. Patent No. 8,332,844

TABLE OF CONTENTS

EXHIBIT LIST i

TABLE OF ABBREVIATIONS ii

I. ASSIGNMENT 1

II. BACKGROUND AND QUALIFICATIONS 2

III. MATERIALS CONSIDERED 12

IV. LEVEL OF ORDINARY SKILL IN THE ART 13

V. RELEVANT LEGAL STANDARDS 14

VI. SUMMARY OF OPINIONS 18

VII. TECHNOLOGY OVERVIEW 18

A. General Computer Operations 19

1. Networking Overview 20

B. Data Storage Devices and File Systems 25

1. Files and File Systems 26

C. Networked and Distributed Storage Systems 30

D. Caching Technologies 34

1. Least-Recently Used (LRU) 42

2. Least-Frequently Used (LFU) 42

E. Copy-on-Write and Snapshots 44

F. Indexing 56

G. Virtualization 57

VIII. THE ’844 PATENT 61

A. Overview 61

B. The ’844 Patent Discloses Indexing at the Block Layer....71
C. The ’844 Patent’s Union Block Device (UBD)....74
D. Prosecution History....76
1. The ’477 Application....76
2. The ’622 Application....83
3. Priority Date....84
IX. THE CLAIMS....85
X. OVERVIEW OF THE PRIOR ART....90
A. Menage....91
B. Murphy....102
C. Birse....110
D. Rothman....120
XI. CLAIM CONSTRUCTION....123
XII. DETAILED EXPLANATION OF GROUNDS....124
A. Ground 1: Menage Renders Claims 1-13 Obvious....124
1. Claim 1....124
2. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value....140
3. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance....143
4. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said

first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.....145

5. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache. 147

6. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only. 149

7. Claim 7 149

8. Claim 8: The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and providing said first portion of said data to said at least one of said compute nodes from said cache memory. 150

9. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request. 151

10. Claim 10: The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. 152

11. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments. 153

12. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory. 153

13. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only. 153

B. Ground 2: Menage in View of Murphy Renders Claims 14-27 Obvious 153

1. Motivation to Combine Menage with Murphy 153

2. Claim 14 157

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance. 163

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 163

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache. 163

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only. 164

7. Claim 19 164

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit. 165

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units. 166

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only. 166

11. Claim 23 167

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node. 169

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.....169

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.....170

15. Claim 27: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.....171

C. Ground 3: Birse in View of Rothman Renders Claims 1-13 Obvious.....172

1. Motivation to Combine Birse with Rothman 172

2. Claim 1 176

3. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value. 195

4. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance. 196

5. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.....197

6. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache. 200

7. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only. 201

8. Claim 7 202

9. Claim 8: The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and providing said first portion of said data to said at least one of said compute nodes from said cache memory. 203

10. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request. 204

11. Claim 10: The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value. 205

12. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments. 206

13. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory. 206

14. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only. 206

D. Ground 4: Birse in View of Murphy Renders Claims 14-27 Obvious 206

1. Motivation to Combine Birse with Murphy 206

2. Claim 14 210

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance. 215

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices con figured to distribute application

environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images. 216

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache. 216

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only. 216

7. Claim 19 216

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit. 217

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units. 219

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only. 219

11. Claim 23 219

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node. 222

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion 222

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion. 223

15. Claim 27: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion. 224

XIII. SECONDARY CONSIDERATIONS OF NON-OBVIOUSNESS 224

XIV. CONCLUSION 224

EXHIBIT LIST

Exhibit	Description
1001	U.S. Patent No. 8,332,844
1002	File History for U.S. Patent No. 8,332,844
1004	U.S. Patent No. 6,618,736
1005	U.S. Patent No. 7,395,324
1006	U.S. Patent No. 7,089,300
1007	U.S. Patent No. 7,398,382
1008	File History for U.S. Patent Appl. No. 11/395,816
1009	A. Silberschatz & P. B. Galvin, *Operating System Concepts* (4th ed. 1994)
1010	D. P. Bovet & M. Cesati, *Understanding the Linux Kernel* (1st ed. 2000)
1011	U.S. Patent No. 5,313,646
1012	File History for U.S. Patent Appl. No. 11/026,622
1013	Excerpts from Microsoft Computer Dictionary (3rd ed. 1997)
1014	Internet Small Computer Systems Interface (iSCSI), RFC 3720, IETF (April 2004), https://www.rfc-editor.org/rfc/pdfrfc/rfc3720.txt.pdf
1015	Dave Hitz *et al.*, Network Appliance Inc., *File System Design for an NFS File Server Appliance*, USENIX 1994
1016	U.S. Patent No. 7,334,095
1017	Hugo Patterson *et al.*, Network Appliance Inc., *File System Based Asynchronous Mirroring for Disaster Recovery*, USENIX 2002
1018	U.S. Patent No. 6,668,264
1019	File History for U.S. Patent Appl. No. 11/395,816

TABLE OF ABBREVIATIONS

Abbreviation	Term
’477 application	U.S. Patent Appl. No. 11/709,477
’816 application	U.S. Patent Appl. No. 11/395,816
’622 application	U.S. Patent Appl. No. 11/026,622
’844 patent	U.S. Patent No. 8,332,844 (EX1001)
AIA	America Invents Act
Birse	U.S. Patent No. 7,089,300 (EX1006)
Challenged Claims	Claims 1–27 of U.S. Patent No. 8,332,844
Fair	U.S. Patent No. 7,334,095 (EX1016)
Fig.	Figure
Hitz	Dave Hitz *et al.*, *File System Design for an NFS File Server Appliance Network Appliance*, USENIX 1994 (EX1015)
IPR	*inter partes* review
Menage	U.S. Patent No. 6,618,736 (EX1004)
Murphy	U.S. Patent No. 7,395,324 (EX1005)
Patent Owner	Intellectual Ventures II LLC
Patterson I	Hugo Patterson *et al.*, Network Appliance Inc., *File System Based Asynchronous Mirroring for Disaster Recovery*, USENIX 2002 (EX1017)
Patterson II	U.S. Patent No. 6,668,264 (EX1018)
Petitioner	Docker Inc.
POSITA	Person[s] of ordinary skill in the art
Rothman	U.S. Patent No. 7,398,382 (EX1007)

I, Erez Zadok, Ph.D., declare that:

I. ASSIGNMENT

1. I have been retained by Docker Inc. ("Petitioner") as an independent expert consultant in this proceeding before the United States Patent and Trademark Office ("PTO").

2. My consulting company, Zadoks Consulting Services, is being compensated for my time at my standard consulting rate. I am also being reimbursed for expenses that I may incur during the course of this work.

3. My compensation is in no way contingent on the nature of my findings, the presentation of my findings in testimony, or the outcome of this or any other proceeding. I have no other interest in this proceeding.

4. I have been asked to consider whether certain references disclose or suggest the features recited in the claims of U.S. Patent No. 8,332,844 ("the '844 patent") (EX1001).[1] My opinions are set forth below.

[1] Where appropriate, I refer to the exhibits enumerated in the "Exhibit List" above, which I understand will be attached to the petition for *inter partes* review of the '844 patent (the "Petition"). All emphasis in quoted portions of the exhibits has been added, unless otherwise noted.

II. BACKGROUND AND QUALIFICATIONS

5. I am a Professor in the Computer Science Department at Stony Brook University (part of the State University of New York ("SUNY") system). I direct the File-systems and Storage Lab (FSL) at Stony Brook's Computer Science Department. My research interests include file systems and storage systems, operating systems, transactional systems including database technologies, information technology and system administration, security/privacy and information assurance, networking, energy efficiency, performance and benchmarking, virtualization, cloud systems, compilers, applied machine learning, and software engineering.

6. I studied at a professional high school in Israel, focusing on electrical engineering ("EE"), and graduated in 1982. I spent one more year at the high school's college division, receiving a special Certified Technician's degree in EE. I then went on to serve in the Israeli Defense Forces for three years (1983–1986). I received my Bachelor of Science degree in computer science ("CS") in 1991, my master's degree in CS in 1994, and my Ph.D. in CS in 2001—all from Columbia University in New York.

7. When I began my undergraduate studies at Columbia University, I also started working as a student assistant in the various campus-wide computer labs, eventually becoming an assistant to the head labs manager, who was managing all

public computer labs on campus. During that time, I also became more involved with research within the CS Department at Columbia University, conducting research on operating systems, file and storage systems, distributed and networked systems, security, and other topics. I also assisted the CS department's computer administrators in managing the department's computers, which included storage, IT, networking, and cyber-security related duties.

8. In 1991, I joined Columbia University's CS department as a full-time systems administrator, studying towards my MS degree part-time. My MS thesis topic is related to file system reliability, fault tolerance, replication, and failover in mobile networked storage systems using file virtualization. My main duties as a systems administrator involved installing, configuring, and managing many networked servers, proxies, and desktops running several operating systems, as well as network devices setup; this included many software and hardware upgrades, device upgrades, and BIOS firmware/chipset updates/upgrades. My duties also included ensuring reliable, secure, authenticated access to networked systems/storage and licensed software, as well as software updates, security and bug fixes. Examples of servers and their protocols included email (SMTP), file transfer (FTP), domain names (DNS), network file systems (NFS), network news systems (NNTP), and Web (HTTP).

9. In 1994, I left my systems administrator position to pursue my doctoral studies at Columbia University. My PhD thesis topic was on versatile file system development using stackable (virtualized) file systems, with examples in the fields of security and encryption, efficiency, reliability, and failover. I continued to work part-time as a systems administrator at the CS department, and eventually I was asked to serve as manager to the entire information technology ("IT") staff. From 1991 to 2001, I was also a member of the faculty-level Facilities Committee that oversaw all IT operations at the CS department.

10. As part of my PhD studies at Columbia, I collaborated on projects to develop advanced AI-like techniques to detect previously unknown viruses (a.k.a. "zero-day malware"), using data mining and rule-based detection. This work led to several highly cited papers (over 1,600 citations for one of the papers alone) and two patents. I also became a Teaching Assistant ("TA") for a first-ever Computer Security course given at Columbia University's CS department with Dr. Matt Blaze as instructor.

11. From 1990 to 1998, I consulted for SOS Corporation and HydraWEB Technologies as a systems administrator and programmer, managing data storage use and backup/restore duties, databases, web servers, as well as information assurance and cyber-security (*e.g.*, malware protection, software licensing). From 1994 to 2000, I led projects at HydraWEB Technologies, and then became the Director of

Software Development—overseeing the development of several products and appliances such as stateful firewalls and HTTP load-balancers, utilizing network-virtualization and high-availability techniques. From 2009 to 2019, I have consulted for Packet General Networks, a startup specializing in secure, virtualized, network storage and applications' data security in the cloud.

12. In 2001, I joined the faculty of Stony Brook University, a position I have held since that time. In 2002, I joined the Operations Committee, which oversees the IT operations of the CS department at Stony Brook University. From 2006 to 2010, I was the Director of IT Operations of the CS department. My day-to-day duties included setting policies regarding computing, hiring and training new staff, assisting any staff with topics of my specialty, defining requirements for new software/hardware, and purchasing. From 2010 to 2015, I had served as the Co-Chair to the Operations Committee. From 2016 to 2019, I oversaw the IT Operations as the Chair of the Operations Committee. A significant component of these duties included defining and helping implement policies for data management, to ensure the security of users and their data, and data reliability and availability, while minimizing the inconvenience and performance impact to users. I personally helped setup and maintain an initial virtual-host infrastructure in the department. Since late 2019, I've been a member of the department's Executive Committee that also oversees all IT operations.

13. In 2017, I became the department's Graduate Academic Adviser, advising all Master students (over 400 annually on average) and many other graduate students on an assortment of academic matters. In August 2024, I took over as the department's Graduate Program Director, overseeing the entire graduate CS program (700-800 students annually on average).

14. Since 2001, I have personally configured and managed my own research lab's network. This includes setting up and configuring multiple storage systems (*e.g.*, NFS, CIFS/SMB, NAS), virtual and physical environments, applications such as database (*e.g.*, MySQL, Postgresql), Web servers (*e.g.*, Apache), and mail servers; user access control (*e.g.*, NIS, LDAP), backups and restores, snapshot policies, and more. I've personally installed, configured, changed, replaced parts, and upgraded components in numerous devices including mobile devices, laptops, desktops, and servers, both physical and virtual.

15. Since 1995, I have taught courses on operating systems, storage and file systems, advanced systems programming in Unix/C, systems administration, data structures, data/software security, and more. My courses often use storage, file systems, distributed systems, and system/network security as key teaching principles and practical examples for assignments and projects. I have taught these concepts and techniques to my students, both to my direct advisees as well as in my courses. For example, in my graduate Operating Systems course, I often cover Linux's kernel

mechanisms to protect users, applications, and data files, virtual file systems, as well as distributed storage systems (*e.g.*, NFS). And in the System Administration undergraduate course, I covered many topics such as networking, storage, backups, and configuring complex applications such as mail, web, and database servers.

16. My research often investigates computer systems from many angles: security, efficiency, energy use, scalability, reliability, portability, survivability, usability, ease-of-use, versatility, flexibility, and more. My research gives special attention to balancing five often- conflicting aspects of computer systems: performance, reliability, energy use, security, and ease-of-use.

17. Since joining Stony Brook University in 2001, my group in the File-systems and Storage Lab ("FSL") has developed many file systems and operating system extensions; examples include a highly-secure cryptographic file system, a portable copy-on-write ("COW") versioning file system, a tracing file system useful to detect intrusions, a replaying file system useful for forensics, a snapshotting and sandboxing file system, a namespace unification file system (that uses stackable, virtualized, file-based COW), an anti-virus file system, an integrity-checking file system, a load balancing and replication/mirroring file system, network file system extensions for security and performance, distributed secure cloud-based storage systems, transactional key-value stores and file systems, OS-level embedded databases, a compiler to convert user-level C code to in-kernel efficient yet safe

code, GCC plugins, stackable file system templates, and a Web-based backup system. Many of these projects used one form of virtualization or another (storage, network, host, etc.). I continue to maintain and release newer versions of some of these file systems and software.

18. I have published over 120 refereed publications (in ACM, IEEE, USENIX, and more). To date, my publications have been cited more than 10,000 times (as per Google Scholar as of April 16, 2025). My papers cover a wide range of related technologies such as file systems, storage systems, transactional systems, security, clouds and virtualization, performance benchmarking and optimization, energy efficiency, system administration, web systems, and more. I also published a book titled "Linux NFS and Automounter Administration" (Sybex, 2001), covering systems administration topics related to network storage and data security.

19. Some of my research has led to public software releases that have been used worldwide. I have publicly maintained the Amd Berkeley Automounter in a package called "am-utils" since 1992; this software helps administrators manage the multitude of file system mounts on dozens of different Unix systems, especially helping to automate access to multiple NFS/NAS storage volumes. Since 1997, I have maintained and released several stackable (virtualized) file system software projects for Linux, FreeBSD, and/or Sun Solaris, in a package called FiST. One of my stackable file system encryption projects, called Cryptfs, became the basis for

IBM's public release of eCryptfs, now part of Linux. Packet General Networks, for whom I have provided consulting services between 2009 and 2019, licensed another encryption file system called Ncryptfs. Another popular file system released in 2003, called Unionfs, offers virtual namespace unification, transparent shadow copying (a.k.a. copy-on-write or COW), file system snapshotting (*e.g.*, useful for forensics and disaster recovery), and the ability to save disk space by sharing a read-only copy of data among several computers, among other features.

20. My research and teaching make extensive use of data security features. For example, each time I taught the graduate operating system course, the first homework assignment includes the creation of a new system call that performs new or added functionality, often for encrypting a file or verifying its integrity; many of my other assignments cover topics of user/process access control, anti-virus filtering, and more. Since 2001, over 1,000 graduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

21. Moreover, in an undergraduate course titled "Advanced Systems Programming in Unix/C," I cover many topics of system security and vulnerabilities, such as the structure of UNIX processes, and memory segments such as the heap and stack. This course covers details of several hundred Linux system calls. Often, the first assignment for this course is to develop a tool to encrypt/decrypt files using advanced ciphers, use digital signatures to certify the cipher keys used, and reliably

recover files in case of failures. Since 2001, several hundred undergraduate students were exposed to these principles directly through my teaching and research at Stony Brook University.

22. In another undergraduate course, System Administration, I taught network configuration, security, and storage configuration and reliability. In a special topics course on Storage Systems, I covered many topics such as data deduplication, RAID, transactional storage, storage hardware including modern Flash based ones, virtual storage, backup/restore, snapshots and continuous data protection ("CDP"), NAS and SAN, and NFS.

23. Overall, in addition to the aforementioned experience, my technical experience relevant to the '844 patent at the time of the alleged invention included the configuration, use, research, development, and/or programming of the following technologies and systems: data storage devices, drivers, and services; operating systems layers including (virtual) block layer, file systems, and virtual file systems; copy-on-write storage system and file systems (e.g., Unionfs), including snapshots; commercial and open-source storage appliances (including from Netapp); and caching techniques in one or multi-tier systems.

24. My research has been supported by many federal and state grants as well as industry awards, including an NSF CAREER award, two IBM Faculty awards, two NetApp Faculty awards, a Western Digital award, a Facebook award, several Dell-

EMC awards, and several equipment gifts. I received the 2008 SUNY Chancellor's Excellence in Teaching award, and the 2022 SUNY Chancellor's Award for Excellence in Scholarship and Creative Activities (both awards can be given only once in a lifetime). In 2021, I was named an ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

25. My service record to the community includes serving as the co-chair for the USENIX Annual Technical Conference in 2020 (ATC'20); serving as the co-chair for USENIX File and Storage Technologies (FAST'15) in 2015 and on the FAST Conference Steering Committee from 2015 to 2023; serving on the ACM HotStorage Steering Committee since 2021; and serving as the co-chair in 2012 and on the Steering Committee of the ACM SYSTOR conference since 2012. I have served as an Associate Editor to the ACM Transactions on Storage ("TOS") journal from 2009 to 2022; in 2022, I was named the Editor-in-Chief for ACM's TOS journal.

26. I am a named inventor on four patents, two titled "Systems and Methods for Detection of New Malicious Executables" (U.S. Patent No. 7,487,544, issued February 3, 2009; and U.S. Patent No. 7,979,907, issued July 12, 2011); and two more titled "Multi-Tier Caching," (U.S. Patent No. 9,355,109, issued May 31, 2016; and U.S. Patent 9,959,279, issued May 1, 2018).

27. I have been disclosed as a testifying expert in 19 cases (including *inter partes* review ("IPR") proceedings) in the past four years. I have been deposed 13 times and testified in trial twice.

28. A copy of my *curriculum vitae* is attached as Appendix A.

III. MATERIALS CONSIDERED

29. The opinions contained in this declaration are based on the documents I reviewed, my professional judgment, as well as my education, experience, and knowledge regarding electrical engineering, computer science, video and audio and communications sciences and technologies.

30. In forming my opinions expressed in this declaration, I reviewed the materials enumerated in the "Exhibit List" above.

31. Other materials I refer to in this declaration in support of my opinions.

32. All of the opinions contained in this declaration are based on the documents I reviewed and my knowledge, experience, and professional judgment. My opinions have also been guided by my appreciation of how a person of ordinary skill in the art would have understood the claims and the specification of the '844 patent at the time of the alleged invention, which I have been asked to initially consider as no earlier than December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1012). My opinions reflect how one of ordinary skill in

the art would have understood the '844 patent, the prior art to the patent, and the state of the art at the time of the alleged invention.

33. Based on my experience and expertise, it is my opinion that certain references disclose and/or suggest all the features recited in claims 1-27 ("Challenged Claims") of the '844 patent, as I discuss in detail below.

IV. LEVEL OF ORDINARY SKILL IN THE ART

34. I have been informed and understand that, in the context of an invalidity analysis, a person having ordinary skill in the art ("POSITA") is not a specific real individual, but a hypothetical person who looks to prior art at the time of the invention. I further understand that the factors that may be considered in determining the level of ordinary skill include: (1) the level of education and experience of persons working in the field; (2) the types of problems encountered in the field; (3) the teachings of the prior art regarding solutions to such problems; (4) the sophistication of the technology at the time of the alleged invention; and (5) the rapidity of innovation. I understand that these factors need not all be considered for the analysis, and that one or more of these factors may control.

35. I was asked to provide my opinion on the level of one of ordinary skill in the art with respect to the alleged invention of the '844 patent no earlier than December 30, 2004. In my opinion, a POSITA would have had a bachelor's degree in computer science, computer engineering, or an equivalent degree. A POSITA

would also have had approximately two years of experience working in the fields of networked data storage, file systems, and operating systems. Additional experience might substitute for less education and *vice versa*. It is also my opinion that these qualifications would have been the same on March 30, 2006, and February 21, 2007.

36. As of December 30, 2004, I met, and in fact exceeded, the qualifications of a person of ordinary skill in the art. To be clear, all of my opinions in this declaration are from the perspective of one of ordinary skill in the art as I have defined it here during the relevant timeframe.

V. RELEVANT LEGAL STANDARDS

37. I am not an attorney and offer no legal opinions, but in the course of my work I have had experience studying and analyzing patents and patent claims from the perspective of a person skilled in the art.

38. For the purposes of my analysis and conclusions expressed in this declaration, I have applied the legal principles described in the following paragraphs.

39. Petitioner's counsel has informed me that for the prior art to inherently disclose a claimed limitation, the prior art need not expressly disclose the limitation, so long as the claimed limitation necessarily flows from a disclosure in the prior art. I also understand that it is acceptable to examine evidence outside the prior art reference (extrinsic evidence) in determining whether a feature, while not expressly discussed in the reference, is necessarily present in that reference.

40. Petitioners' counsel has informed me that a patent claim can be considered to have been obvious to a POSITA at the time the application was filed. I am informed that this means that, even if all of the requirements of a claim are not found in a single prior art reference, the claim is not patentable if the differences between the subject matter in the prior art and the subject matter in the claim would have been obvious to a POSITA at the time of the invention.

41. I have been informed by Petitioner's counsel that a determination of whether a claim would have been obvious should be based upon several factors, including, among others:

- The level of ordinary skill in the art at the time the application was filed;
- The scope and content of the prior art; and
- What differences, if any, existed between the claimed invention and the prior art.

42. I have been informed by Petitioner's counsel that a single reference can render a patent claim obvious if any differences between that reference and the claims would have been obvious to a POSITA. Alternatively, I understand that the teachings of two or more references may be combined in the same way as disclosed in the claims, if such a combination would have been obvious to one having ordinary skill in the art. In determining whether a combination based on either a single

reference or multiple references would have been obvious, it is appropriate to consider, among other factors:

- Whether the teachings of the prior art references disclose known concepts combined in familiar ways, and when combined, would yield predictable results;
- Whether a person having ordinary skill in the art could implement a predictable variation, and would see the benefit of doing so;
- Whether the claimed elements represent one of a limited number of known design choices, and would have a reasonable expectation of success by those skilled in the art;
- Whether a person having ordinary skill would have recognized a reason to combine known elements in the manner described in the claim;
- Whether there is some teaching or suggestion in the prior art to make the modification or combination of elements claimed in the patent; and
- Whether the innovation applies a known technique that had been used to improve a similar device or method in a similar way.

43. I have been informed by Petitioner's counsel that one of ordinary skill in the art has ordinary creativity and is not an automaton. Petitioners' counsel has also informed me that in considering obviousness, obviousness may not be

determined using the benefit of hindsight, including hindsight derived from the patent being considered.

44. I further understand that a hypothetical POSITA is assumed to know and to have all relevant prior art in the field of endeavor covered by the patent at issue. I have been informed that there are at least two criteria for determining whether prior art is analogous and thus can be considered prior art: (1) whether the art is from the same field of endeavor, regardless of the problem addressed; and (2) if the reference is not within the field of the patentee's endeavor, whether the reference still is reasonably pertinent to the particular problem with which the patentee is involved. I have also been informed that the field of endeavor of a patent is not limited to the specific point of novelty, the narrowest possible conception of the field, or the particular focus within a given field. I have also been informed that a reference is reasonably pertinent if, even though it may be in a different field from that of the patentee's endeavor, it is one which, because of the matter with which it deals, logically would have commended itself to a patentee's attention in considering their problem.

45. I further understand that, in making a determination as to whether the claimed invention would have been obvious to a POSITA, the U.S. Patent Trial and Appeal Board may consider certain objective factors if they are present, such as: commercial success of products practicing the claimed invention; long-felt but

unsolved need; teaching away; unexpected results; copying; and praise by others in the field. These factors are generally referred to as "secondary considerations" or "objective indicia" of nonobviousness. I understand, however, that for such objective evidence to be relevant to the obviousness of a claim, there must be a causal relationship (called a "nexus") between the claim and the evidence, and that this nexus must be based on a novel element of the claim rather than something in the prior art. I also understand that even when they are present, secondary considerations may be unable to overcome primary evidence of obviousness (such as motivation to combine with predictable results) that is sufficiently strong.

VI. SUMMARY OF OPINIONS

46. This declaration explains the conclusions I have formed based on my analysis. These conclusions are summarized in the table below:

Ground	Basis	Relied-Upon References	Claim(s)
1	Obviousness	Menage	1-13
2	Obviousness	Menage and Murphy	14-27
3	Obviousness	Birse and Rothman	1-13
4	Obviousness	Birse and Murphy	14-27

VII. TECHNOLOGY OVERVIEW

47. In this section, I provide an overview of the background art as understood by a POSITA during the relevant timeframe.

A. General Computer Operations

48. Computers are hardware and software devices that execute instructions. The hardware often includes a central processing unit ("CPU") or other microprocessors, typically called "processors," volatile memory (*e.g.*, RAM (random-access memory), or DRAM (dynamic random-access memory)), non-volatile (persistent) storage (*e.g.*, a hard disk or flash drive), and input/output ("I/O") devices (*e.g.*, network interface card, mouse, keyboard, display). RAM, hard disks, and flash drives are considered, generally, "storage" or "memory" devices because they can store and hold bits of information (whether persistently or not). Computers can have multiple storage devices. Processors are coupled to main memory using a memory bus and coupled to peripherals such as hard disks and network interface cards using one or more I/O busses.

49. Memory and storage devices have considerable differences in their speeds, capacities, and costs. Non-volatile storage devices such as hard disks have the largest capacity, are least expensive per gigabyte, and are slowest (typically operating at millisecond speeds). Volatile DRAM devices are smaller, more expensive, and faster than hard disks (typically operating at microsecond speeds). A CPU's own memory caches and operations are the smallest, most expensive, and fastest (typically operating at nanosecond speeds). Overall, CPUs are about 1,000× faster than DRAM, which in turn is about 1,000× faster than hard disks.

50. Computer programs are often written in a high-level human-readable language (*e.g.*, C, C++, Java, Perl, PHP, Python), then translated using a compiler or script processor to machine instructions understood by the CPU (*e.g.*, Intel or AMD processor). Machine instructions—generally called "software"—are stored in files on persistent media (*e.g.*, HDD or ROM), loaded into DRAM, and then loaded into the CPU where they can be executed. All modern computers operate in this manner, regardless of their purpose: a small laptop, a server in a data center, a mainframe computer, a personal workstation, a proxy device, a gateway, a firewall, a router, an appliance, a virtual machine, a handheld or mobile device, a smart card, a SD card, etc. Computer systems may run multiple software programs, from background servers and daemons to user applications.

51. Users (including administrators) can login to one or more computers (*e.g.*, desktop, laptop, server) and execute one or more applications (*e.g.*, word processing, web browsing, backup tool). Some computers offer external interfaces and APIs to communicate with certain devices and especially peripherals. For example, both a USB storage device as well as a printer use a well- defined "protocol" to allow users to communicate with them through the main computer.

1. Networking Overview

52. A Local Area Network ("LAN") is a typically a network at a local site where network latencies are short and physical distances between computers are

small. A LAN and its computers are often protected from the rest of the Internet using a firewall. EX1013 (Microsoft Computer Dictionary) at 276 ("**LAN** *n*. Acronym for **l**ocal **a**rea **n**etwork. A group of computers and other devices dispersed over a relatively limited area and connected by a communications link that enables any device to interact with any other on the network. LANs commonly include microcomputers and shared resources such as laser printers and large hard disks. The devices on a LAN are known as nodes, and the nodes are connected by cables through which messages are transmitted. *See also* baseband network, broadband network, bus network, collision detection, communications protocol, contention, CSMA/CD, network, ring network, star network, token bus network, token passing, token ring network. *Compare* wide area network.").

53. Another form of a local area network is one that operates inside a single computer. A "localhost" is "[t]he name that is used to represent the same computer on which a TCP/IP message originates. An IP packet sent to localhost has the IP address 127.0.0.1 and does not actually go out to the Internet. *See also* IP address, packet (definition 1), TCP/IP." EX1013 (Microsoft Computer Dictionary) at 287. Often, when users/processes want to communicate with other users/processes, using TCP/IP communications on the same machine, they can use the localhost address: it is considered a trusted, high-speed internal network.

54. Conversely, a Wide-Area Network ("WAN") refers to computers across the entire Internet, which are often distant from each other, exhibit longer latencies, and afford little protection. EX1013 (Microsoft Computer Dictionary) at 507 ("**wide area network** *n*. A communications network that connects geographically separated areas. *Acronym*: WAN.").

55. A firewall typically intercepts all traffic between computers in a LAN and the WAN, monitors and controls access, and can filter out undesired traffic. Often, LAN computers are far more trusted than WAN computers; therefore, a firewall can be said to separate the trusted network (LAN or localhost) from the untrusted network (WAN). EX1013 (Microsoft Computer Dictionary) at 197 ("**firewall** *n*. A security system intended to protect an organization's network against external threats, such as hackers, coming from another network, such as the Internet firewall prevents computers in the organization's network from communicating directly with computers external to the network and vice versa. Instead, all communication is routed through a proxy server outside of the organization's network, and the proxy server decides whether it is safe to let a particular message or file pass through to the organization's network.").

56. A server is a computer or software that provides a service to other computers, called clients. There are many servers on the Internet, for example, Web servers and Mail servers. EX1013 (Microsoft Computer Dictionary) at 430 ("**server**

n. 1. On a local area network (LAN), a computer running administrative software that controls access to the network and its resources, such as printers and disk drives, and provides resources to computers functioning as workstations on the network. 2. On the Internet or other network, a computer or program that responds to commands from a client. For example, a file server may contain an archive of data or program files; when a client submits a request for a file, the server transfers a copy of the file to the client. *See also* client/server architecture. *Compare* client (definition 3)."

57. A client is typically an end-point computer or software, sometimes used by a user, that communicates across LAN and WAN networks; a client may contact servers to request some service. A user's desktop, laptop, and handheld device are examples of clients. (Note that computer can be both a client and a server.) EX1013 (Microsoft Computer Dictionary) at 92 ("**client** *n*. 1. In object-oriented programming, a member of a class (group) that uses the services of another class to which it is not related. *See also* inheritance (definition 1). 2. A process, such as a program or task, that requests a service provided by another program—for example, a word processor that calls on a sort routine built into another program. The client process uses the requested service without having to 'know' any working details about the other program or the service itself. *Compare* child (definition 1), descendant (definition 2). 3. On a local area network or the Internet, a computer

that accesses shared network resources provided by another computer (called a server). *See also* client/server architecture, server.").

58. A client/server architecture describes how client computers interact with server computers. EX1013 (Microsoft Computer Dictionary) at 92 ("**client/server architecture** *n*. An arrangement used on local area networks that makes use of distributed intelligence to treat both the server and the individual workstations as intelligent, programmable devices, thus exploiting the full computing power of each. This is done by splitting the processing of an application between two distinct components: a 'front-end' client and a 'back-end' server. The client component is a complete, stand-alone personal computer (not a 'dumb' terminal), and it offers the user its full range of power and features for running applications. The server component can be a personal computer, a minicomputer, or a mainframe that provides the traditional strengths offered by minicomputers and mainframes in a time-sharing environment: data management, information sharing between clients, and sophisticated network administration and security features. The client and server machines work together to accomplish the processing of the application being used. Not only does this increase the processing power available over older architectures but it also uses that power more efficiently. The client portion of the application is typically optimized for user interaction, whereas the

server portion provides the centralized, multiuser functionality. *See also* distributed intelligence.").

B. Data Storage Devices and File Systems

59. As mentioned in §VII.A above, computers often include two types of storage or memory devices. The first type is a volatile memory device, called Random Access Memory ("RAM"), that is often used to hold executing programs and their data. EX1013 (Microsoft Computer Dictionary), p. 395 ("RAM").

60. The second type is a non-volatile, or persistent storage device, such as a hard disk drive ("HDD"). HDDs are used to store programs and data persistently. EX1011 (Microsoft Computer Dictionary), p. 226 ("hard disk").

61. HDDs are electro-mechanical devices with moving parts. Computers can also have other types of non-volatile storage devices that have no moving parts, also called "solid-state devices," such as flash memories. EX1013 (Microsoft Computer Dictionary), p. 199 ("flash memory").

62. Moreover, any single device (*e.g.*, a hard disk) can be logically broken down into several units called "partitions." EX1011 (Microsoft Computer Dictionary), p. 152 ("**disk partition** *n.* A logical compartment on a physical disk drive. A single disk might have two or more logical disk partitions, each of which would be referenced with a different disk drive name. Multiple partitions are divided into a primary (boot) partition and one or more extended partitions.").

63. Above the storage media often lives a logical software layer called a "file system." File systems provide a convenient abstraction to users and applications of files, folders or directories, and naming. EX1013 (Microsoft Computer Dictionary), p. 196 ("**file system** *n.* In an operating system, the overall structure in which files are named, stored, and organized. A file system consists of files, directories, or folders, and the information needed to locate and access these items. The term can also refer to the portion of an operating system that translates requests for file operations from an application program into low-level, sector-oriented tasks that can be understood by the drivers controlling the disk drives. *See also* driver.").

1. Files and File Systems

64. A file is a collection of data and attributes, or meta-data, about the file, including the file's name and extension. EX1013 (Microsoft Computer Dictionary) at p. 194 ("**file** *n.* A complete, named collection of information, such as a program, a set of data used by a program, or a user-created document. A file is the basic unit of storage that enables a computer to distinguish one set of information from another. A file is the 'glue' that binds a conglomeration of instructions, numbers, words, or images into a coherent unit that a user can retrieve, change, delete, save, or send to an output device.").

65. File attributes are also called file meta-data and include security attributes, the name (title), size of a file, timestamps, and more. EX1013 (Microsoft Computer Dictionary) at p. 305 ("**meta data** or **metadata** *n.* 1. Data about data. For example, the title, subject, author, and size of a file constitute metadata about the file. *See also* data dictionary, repository."), p. 194 ("**file attribute** *n.* A restrictive label attached to a file that describes and regulates its use—for example, hidden, system, read-only, archive, and so forth. In MS-DOS, this information is stored as part of the file's directory entry.").

66. A file's name is an important attribute of a file and conveys a lot of useful information about the file's meaning and purpose. EX1013 (Microsoft Computer Dictionary) at p. 195 ("**filename** *n.* The set of letters, numbers, and allowable symbols assigned to a file to distinguish it from all other files in a particular directory on a disk. A file name is the label under which a computer user saves and requests a block of information. Both programs and data have file names and often extensions that further identify the type or purpose of the file. Naming conventions, such as maximum length and allowable characters of a file name, vary from one operating system to another. *See also* directory, path (definition 5).").

67. A file's path or path-name conveys the location of a file in the name hierarchy and even which file server the file may reside on (*e.g.*, the local C:\ drive or a network drive like N:\). EX1013 (Microsoft Computer Dictionary) at p. 356

("**path** *n.* […] 5. In file storage, the route followed by the operating system through the directories in finding, sorting, and retrieving files on a disk."), *id.* ("**pathname** *n.* In a hierarchical filing system, a listing of the directories or folders that lead from the current directory to a file. *Also called:* directory path.").

68. Moreover, a file's extension conveys vital information about the file's type and hence its meaning and purpose. For example, a file ending with "xls" is known to be a Microsoft Excel spreadsheet, files with "doc" extensions are Microsoft Word files, and those ending with "html" are Web server hyper-text markup language files. EX1013 (Microsoft Computer Dictionary) at p. 194 ("**file extension** *n. See* extension (definition 1)."), p. 195 ("**filename extension** *n. See* extension (definition 1)."), p. 186 ("**extension** *n.* 1. A set of characters added to a filename that serves to extend or clarify its meaning or to identify a file as a member of a category. An extension may be assigned by the user or by a program, as, for example, .com or .exe for executable programs that MS-DOS can load and run.").

69. The (usually non-volatile) storage media that holds the actual bits of file data is often called the "secondary storage" or the "backing storage" (because it is in the "back end"); this is different than "backup storage" that's often used to backup files for disaster recovery purposes. A "primary storage" is "Random access memory (RAM); the main general-purpose storage region to which the microprocessor has direct access. A computer's other storage options, such as disks

and tape, are called *secondary storage* or (sometimes) *backing storage*." EX1013 (Microsoft Computer Dictionary), pp. 378-379. A "secondary storage" is "Any data storage medium other than a computer's random access memory (RAM)-typically tape or disk. *Compare* primary storage." EX1013 (Microsoft Computer Dictionary), p. 425.

70. Above the storage media often lives a logical software layer called a "file system." File systems provide a convenient abstraction to users and applications of files, folders or directories, and naming. EX1013 (Microsoft Computer Dictionary), p. 196 ("**file system** *n.* In an operating system, the overall structure in which files are named, stored, and organized. A file system consists of files, directories, and the information needed to locate and access these items. The term can also refer to the portion of an operating system that translates requests for file operations from an application program into low-level, sector-oriented tasks that can be understood by the drivers controlling the disk drives. *See also* driver.").

71. As an example, the "FAT file system" is "[t]he system used by MS-DOS to organize and manage files. The FAT (file allocation table) is a data structure that MS-DOS creates on the disk when the disk is formatted. When MS-DOS stores a file on a formatted disk, the operating system places information about the stored file in the FAT so that MS-DOS can retrieve the file later when requested. The FAT is the only file system MS-DOS can use; OS/2, Windows NT, and Windows 95

operating systems can use the FAT file system in addition to their own file systems (HPFS, NTFS, and VFAT, respectively). *See also* file allocation table, HPFS, NTFS, OS/2, VFAT, Windows 95, Windows NT."). EX1013 (Microsoft Computer Dictionary) at 189-190.

72. Finally, before any storage media or partition can be used to access files and folders, it must be formatted. To "format" is "2. To prepare a disk for use by organizing its storage space into a collection of data 'compartments,' each of which can be located by the operating system so that data can be sorted and retrieved. When a previously used disk is formatted, any preexisting information on it is lost." EX1013 (Microsoft Computer Dictionary), p. 204 ("Format" definition 2).

C. Networked and Distributed Storage Systems

73. As discussed in §VII.B above, storage devices provide raw storage to computers. One of the most popular storage device protocols is SCSI, which is an "[a]cronym for **S**mall **C**omputer **S**ystem **I**nterface) a standard high-speed parallel interface defined by the X3T9.2 committee of the American National Standards Institute (ANSI). A SCSI interface is used to connect microcomputers to SCSI peripheral devices, such as 1nany hard disks and printers, and to other computers and local area networks. *Compare* ESDI, IDE." EX1013 (Microsoft Computer Dictionary), p. 423.

74. A "SCSI network" is "[a] set of devices on a SCSI bus, which acts like a local area network. *See also* SCSI." EX1013 (Microsoft Computer Dictionary), p. 424.

75. Indeed, the SCSI protocol was fairly similar to other network protocols, that the IETF began standardizing it in the early 2000s. iSCSI is "a transport protocol for Internet Small Computer Systems Interface (iSCSI) that works on top of TCP. The iSCSI protocol aims to be fully compliant with the standardized SCSI architecture model. … As system interconnects move from the classical bus structure to a network structure, SCSI has to be mapped to network transport protocols. IP networks now meet the performance requirements of fast system interconnects and as such are good candidates to 'carry' SCSI." EX1014 (RFC 3720), p. 1.

76. File systems can appear as local volumes (*e.g.*, the C:\ drive in Windows), but they can also be networked, where the physical storage resides on a remote file server. Network file systems allow files and volumes to be shared and accessible from multiple networked computers. EX1013 (Microsoft Computer Dictionary), p. 330 ("**Network File System** *n.* *See* NFS."), p. 328. ("**NFS** *n.* Acronym for **N**etwork **F**ile **S**ystem. A distributed file system developed by Sun Microsystems, Inc. that allows users of Windows NT and UNIX workstations to access remote files and directories on a network as if they were local. *Acronym*: NFS.").

77. Even storage media can be networked into a large cluster of devices, that can export virtual block devices to individual hosts, called a "storage area network" ("SAN").

78. When the amount of data to access grew large enough, and especially if the data had to be concurrently accessed from multiple users/applications, storage researchers and companies leveraged the principles of distributed systems to improve the performance, resiliency, and fault-tolerance of the storage system.

79. A single computer's performance is limited by the physical maximum speeds of its network, processor, memory, and I/O devices. A distributed system, however, improves overall performance by using multiple computers concurrently to process many requests, thus increasing the aggerate throughput manyfold.

80. A single computer also represents a single point of failure: if that one computer is down, off, or broken into, then the service it offers is unavailable. A distributed system, conversely, improves resiliency and security by ensuring that the failure of any one (or even a few) components do not affect the overall service provided. This principle is called "fault tolerance." EX1013 (Microsoft Computer Dictionary), p. 190 ("**fault tolerance** *n.* The ability [] to respond to a catastrophic event or fault, such as a power outage or a hardware failure, in a way that ensures that no data is lost and any work in progress is not corrupted. [] In a fault-tolerant network, the system has the ability either to continue the system's operation without

loss of data or to shut the system down and restart it, recovering all processing that was in progress when the fault occurred.").

81. NFS is but one example of a networked or distributed file system. Many complex software systems have been designed to operate in a distributed manner, where they are more resilient to failures and scale better than software running on a single node.

82. Generally, a "distributed network" is "[a] network in which processing, storage, and other functions are handled by separate units (nodes) rather than by a single main computer." EX1013 (Microsoft Computer Dictionary), p. 154.

83. The "Distributed Computing Environment" is "[a] set of standards from the Open Group (formerly the Open Software Foundation) for development of distributed applications that can operate on more than one platform. *Acronym*: DCE. *See also* distributed processing." EX1013 (Microsoft Computer Dictionary), pp. 153-154.

84. "[D]istributed processing" is "[a] form of information processing in which work is performed by separate computers linked through a communications network. Distributed processing is usually categorized as either plain distributed processing or true distributed processing. Plain distributed processing shares the workload among computers that can communicate with one another. True distributed processing has separate computers perform different tasks in such a way

that their combined work can contribute to a larger goal. The latter type of processing requires a highly structured environment that allows hardware and software to communicate, share resources, and exchange information freely." EX1013 (Microsoft Computer Dictionary), p. 154.

85. Finally, a "distributed file system" is "[a] file management system in which files may be located on multiple computers connected over a local or wide area network." EX1013 (Microsoft Computer Dictionary), p. 154.

D. Caching Technologies

86. Computers store the 0 and 1 values of bits in memory devices. Memory devices can be volatile (lose data when device power is off) or non-volatile (data is not lost when power is off). There are many storage media in existence: tapes, hard disks, floppies, Flash technologies (*e.g.*, SSDs), optical (*e.g.*, CD-ROM and DVDs), DRAM (*e.g.*, system memory), CPU registers, and more. All these devices can be placed on a specific point or range in a 3-dimensional continuum comprising their (a) capacity, (b) performance, and (c) price. Almost universally, the faster performing devices (*e.g.*, DRAM) are smaller and cost more per unit storage, whereas the larger-capacity devices (*e.g.*, HDD) are slower and cheaper per gigabyte of storage capacity. The difference in speed/cost/capacity among devices is very significant and ranges along at least 9 orders of magnitude—or a billion times bigger, faster, or costlier. *See* §VII.A above.

87. Therefore, "[t]he wide variety of storage systems in a computer system can be organized in a hierarchy (Figure 2.6) according to their speed and their cost. The higher levels are expensive, but are fast. As we move down the hierarchy, the cost per bit decreases, whereas the access time increases. This tradeoff is reasonable; if a given storage system were both faster and less expensive than another — other properties being the same — then there would be no reason to use the slower, more expensive memory." EX1009 (Silberschatz) at 42, Fig. 2.6.

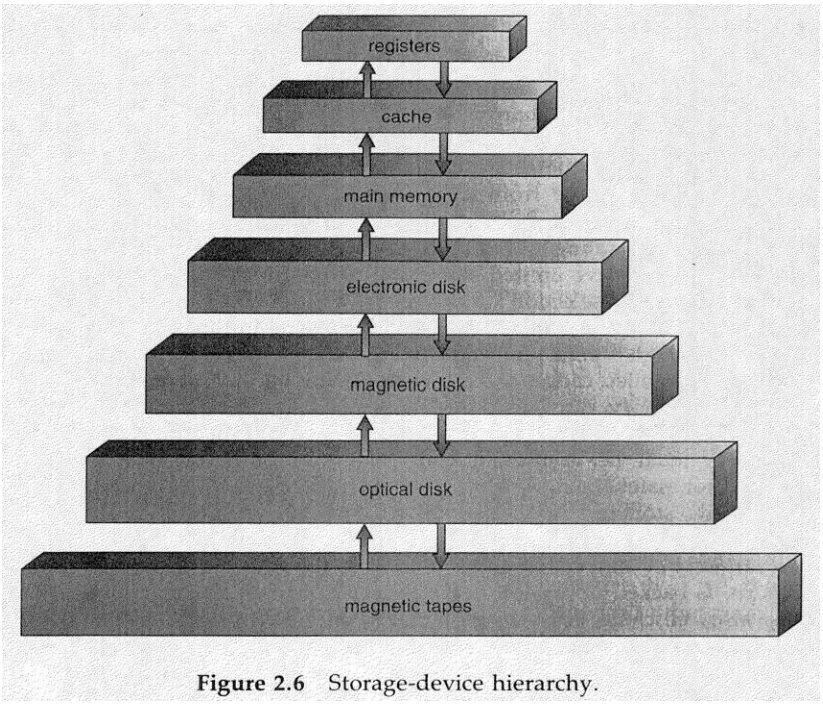


Figure 2.6 Storage-device hierarchy.

88. Managing all these factors is challenging: cost, speed, access speed/latency, size, and volatility. "The design of a complete memory system attempts to balance all these factors: It uses only as much expensive memory as necessary, while providing as much inexpensive, nonvolatile, memory as possible. Caches can be installed to ameliorate performance differences where there is a large

access-time or transfer-rate disparity between two components". EX1009 (Silberschatz) at 42-43.

89. "*Caching* is an important principle of computer systems, in both hardware and software. Information is normally kept in some storage system (such as main memory). As it is used, it is copied into faster storage system, the cache, on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we used the information directly from the cache; if it is not, we use the information from the main storage system, putting a copy in the cache under the assumption that there is a high probability that it will be needed again". EX1009 (Silberschatz) at 43.

90. A "cache" is "[a] special memory subsystem in which frequently used data values are duplicated for quick access. A memory cache stores the contents of frequently accessed RAM locations and the addresses where these data items are stored. When the processor references an address in memory, the cache checks to see whether it holds that address. If it does hold the address, the data is returned to the processor; if it does not, a regular memory access occurs. A cache is useful when RAM accesses are slow compared with the microprocessor speed, because cache memory is always faster than main RAM memory. *See also* disk cache, wait state." EX1013 (Microsoft Computer Dictionary) at 72.

91. Caches can exist at many levels. A "CPU cache" is "[a] section of fast memory linking the central processing unit (CPU) and main memory that temporarily stores data and instructions the CPU needs to execute upcoming commands and programs. Considerably faster than main memory, the CPU cache contains data that is transferred in blocks, thereby speeding execution. The system anticipates the data it will need through algorithms. *Also called* cache memory, memory cache. *See also* cache, central processing unit, VCACHE." EX1011 (Microsoft Computer Dictionary) at 122. CPUs internally can have 2 or more caches, marked "level 1" or L1, "level 2" or L2, etc. *See also* EX1013 (Microsoft Computer Dictionary) at 276 ("L1 cache" and "L2 cache" definitions).

92. A "disk cache" "[a] portion of a computer's random access memory (RAM) set aside for temporarily holding information read from disk. A disk cache does not hold entire files, as does a RAM disk (a portion of memory that acts as if it were a disk drive). Instead, a disk cache is used to hold information that either has recently been requested from disk or has previously been written to disk. If the required information remains in a disk cache, access time is considerably faster than if the program must wait for the disk drive mechanism to fetch the information from disk. *See also* cache. *Compare* disk buffer." EX1013 (Microsoft Computer Dictionary) at 151.

93. A "RAM cache" is "[c]ache memory that is used by the system to store and retrieve data from the RAM. Frequently accessed segments of data may be stored in the cache for quicker access compared with secondary storage devices such as disks. *See also* cache, RAM." EX1013 (Microsoft Computer Dictionary) at 395.

94. In any cache hierarchy, typically the cache (above) is faster and smaller than the backend location (below) from where the data was retrieved. Therefore, caches must be managed appropriately. If a data item is found in a cache it is called a "cache hit"—otherwise it is a "cache miss" that requires retrieving the backend data and copying it into the cache. A "hit" is "1. A successful retrieval of data from a cache rather than from the slower hard disk or RAM. *See also* cache, hard disk, RAM. 2. A successful retrieval of a record matching a query in a database. *See also* query (definition 1), record[1]. 3. Retrieval of a document, such as a home page, from a Web site." EX1013 (Microsoft Computer Dictionary) at 234-235.

95. Cached data is typically stored temporarily. At some point a cache space fills up, and the system must determine which items to "evict"; often the oldest, least used items are evicted first. "Since caches have limited size, *cache management* is an important design problem. Careful selection of the cache size and of a replacement policy can result in 80 to 99 percent of all accesses being in the cache, resulting in extremely high performance. Various replacement algorithms for

software-controlled caches are discussed in Chapter 9." EX1009 (Silberschatz) at 44.

96. Such caching techniques are particularly effective when considering users' work habits and workload patterns. Studies have shown that users and applications often operate on a small set of data files (called a "working set") within a narrow time period. For example, while drafting this very declaration, I am editing the same Word document repeatedly over a period of several days and weeks. Thus, my OS (and any OS prior to 1998) recognizes this access pattern and keeps this Word file cached in DRAM for long time periods. Conversely, it makes less sense for the OS to waste precious DRAM to cache documents I've not worked on for months. However, if, for any reason, I try to access a document I've not opened in a long time, and the document is not already cached in DRAM, then indeed my OS will have to incur a longer latency—and I will have to wait for however long it takes to retrieve the older document from the slower HDD (and cache it into DRAM).

97. Cache management considers when to "admit" data into the cache from backend sources and when to "evict" older data in the cache to make room for new items. One of the most popular and successful cache eviction algorithms is Least-Recently-Used ("LRU"), a heuristic stating that items used more recently are more likely to be used in the near future, and therefore should be kept—while lesser used items should be evicted. LRU uses various thresholds to determine when and what

to evict, based on space available in the cache, the desire to make extra room in the cache for new items, and/or the length of time that items have been in the cache and when were they last used. "LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time." *See* EX1009 (Silberschatz) at 320; *see also, generally*, *id.*, 319-325.

98. Admitting an item from a backend into a cache, or retrieving it from the cache itself, can be seen as a "get" (or read) operation. Conversely, an application inserting an item into a cache, is also called "put" (or write) operation. When new items are inserted into a cache by an application, they are said to be "dirty" because their content is newer than the corresponding backend item. Such dirty items must be "flushed" or written to the backend before they are removed from the cache. A "dirty bit" is "[a] bit used to mark modified data in a cache so that the modifications may be carried over to primary memory. *See also* bit, cache." EX1013 (Microsoft Computer Dictionary) at 149.

99. To "flush" is "[t]o clear a portion of memory. For example, to flush a disk file buffer is to save its contents on disk and then clear the buffer for filling again." EX1013 (Microsoft Computer Dictionary) at 202 (definition 2). In some caches, items in the cache could become "stale" if the corresponding backend item

could have been updated: in that case, some caches attempt to revalidate any cached item before returning it to an application, and as needed remove stale items.

100. Caching is applicable anywhere there is room in one memory device to access data much faster than its source location. Other than RAM-HDD caches, one can cache items on a local storage device that would otherwise take even longer to retrieve than the HDD's access speeds (*e.g.*, network storage, distant servers). A "proxy server" is "[a] firewall component that manages Internet traffic to and from a local area network (LAN) and can provide other features, such as document caching and access control. A proxy server can improve performance by supplying frequently requested data, such as a popular Web page, and can filter and discard requests that the owner does not consider appropriate, such as requests for unauthorized access to proprietary files. *See also* firewall." EX1013 (Microsoft Computer Dictionary) at 387.

101. Silberschatz describes several algorithms used to manage memory resources in an OS. *See* EX1009 (Silberschatz), 315-325 "9.5 Page-Replacement Algorithms," describing techniques such as FIFO, LRU, LFU, MFU, and more. EX1009 (Silberschatz), 315 ("There are many different page-replacement algorithms. Probably every operating system has its own unique replacement scheme. How do we select a particular replacement algorithm? In general, we want the one with the lowest *page-fault rate*.").

1. Least-Recently Used (LRU)

102. One of the oldest, most effective, and best-known algorithms is the Least-Recently- Used (LRU) algorithm. LRU leverages the aforementioned property that users are more likely to work on data files that were in active use in the recent past. LRU orders all cached items by the last time they were accessed. When the cache needs to make room for new items, LRU evicts enough items starting from the least-recently (*e.g.*, oldest) items first. *See* EX1009 (Silberschatz), 319-320 ("**9.5.3 LRU Algorithm** [] If the optimal algorithm is not feasible, perhaps an approximation to the optimal algorithm is possible. … If we use the recent past as an approximation of the near future, then we will replace the page that *has not been used* for the longest period of time (Figure 9.11). This approach is the *least recently used (LRU)* algorithm. LRU replacement associates with each page the time of that page's last use. When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.").

2. Least-Frequently Used (LFU)

103. Another popular algorithm is Least-Frequently-Used ("LFU"). LFU counts not the recency but the frequency of access of cached items. LFU evicts items that have the lowest frequency of use first. EX1009 (Silberschatz), 325 ("**LFU Algorithm**: The *least frequently used (LFU)* page-replacement algorithm

requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count.").

104. Neither LRU nor LFU are perfect. Both work well for certain access patterns but may make poor decisions for other access patterns, possibly evicting items that are needed shortly after eviction. For that reason, researchers have developed cache-management techniques that consider other properties and thresholds: the size of the cached object, object priorities, who are the users who requested the item, inter-relationship among items (*e.g.*, items A and B are needed together), and numerous permutations and combinations (*e.g.*, LRU+LFU) over many years, starting in the 1970s. In sum, all cache management techniques attempt to prioritize cached data—which data should be evicted (because it is less important) and which should be kept (because it is deemed more important).

105. While these techniques are described here in the context of cache management, they are generally applicable anywhere resources are managed. When resources are managed, resource consumers are often prioritized. Such priorities are vital to determine which resources to move elsewhere, while minimizing disruption to operations; it is obvious that, to minimize disruption, resource consumers with the lowest priority should be removed (or discarded) first. For example, one can use LRU, LFU, etc. to prioritize resource users and decide which applications or VMs to migrate from one host (or data center) to another, which network packets to

transmit first, which applications to schedule to run first, which web pages to cache locally, etc.

106. And once again, poor priorities and "predictions" can result in less favorable outcomes. For example, a VM that wasn't deemed a high priority before could be offloaded to a distant data center with fewer resources; but if that VM (or the applications it runs) suddenly becomes more important (*e.g.*, due to a spike of activity), then that VM and its application will suffer poor performance, whereas if that VM was not migrated in the first place, it would have performed much better now.

E. Copy-on-Write and Snapshots

107. CoW has been used in volatile memory systems inside most operating systems. But CoW can be used elsewhere. Most responsible users would make periodic backups of their data files, often on a daily schedule. Backups are therefore a point-in-time copy of users' data files. Network Appliance (NetApp), a company producing storage appliances, popularized the idea of using CoW inside storage systems. Netapp called these point-in-time copies "snapshots." Netapp storage appliances (which I've used and maintained for nearly 20 years, *see* §II above) can periodically take read-only snapshots of the storage appliance, thus producing a point-in-time read- only image of the storage system. Snapshots can be taken more frequently (*e.g.*, every hour or even few minutes). Snapshots also save on disk

space, because data blocks files/sectors that have no changed blocks between snapshots, are shared across multiple snapshots; this is a form of data "de-duplication" that can save a lot of space compared to, say, taking full (and even incremental file-based) backups periodically.

108. Netapp's appliances perform CoW and manage snapshots at the (disk) block layer, but they include a full-fledged file system layer on top of the block layer. That file system can be accessible over the network using NFS (for Unix hosts) or CIFS/Samba (for Windows hosts). *See* §VII.B.1 above. Netapp's CoW engine/driver therefore resides between the file system layer and the actual storage devices.

109. Netapp designed a file system optimized for snapshots and network access, using CoW. "WAFL (Write Anywhere File Layout) … is a file system designed specifically to work in an NFS appliance. The primary focus is on the algorithms and data structures that WAFL uses to implement Snapshots™, which are read-only clones of the active file system. WAFL uses a copy- on-write technique to minimize the disk space that Snapshots consume." EX1015 ("Hitz") at p. 4.

110. Snapshots can be created and deleted as needed, offering space saving and improved reliability. "WAFL's primary distinguishing characteristic is Snapshots, which are read- only copies of the entire file system. WAFL creates and deletes Snapshots automatically at prescheduled times, and it keeps up to 20

Snapshots on-line at once to provide easy access to old versions of files. Snapshots use a copy-on-write technique to avoid duplicating disk blocks that are the same in a Snapshot as in the active file system. Only when blocks in the active file system are modified or removed do Snapshots containing those blocks begin to consume disk space. Users can access Snapshots through NFS to recover files that they have accidentally changed or removed, and system administrators can use Snapshots to create backups safely from a running system. In addition, WAFL uses Snapshots internally so that it can restart quickly even after an unclean system shutdown." EX1015 ("Hitz") at p. 6.

111. The FAServer has commands to let system administrators create and delete Snapshots, but it creates and deletes most Snapshots automatically." EX1015 ("Hitz") at p. 7.

112. Netapp maintains meta-data inside files as well as allocation bitmaps. "**3.2. Meta- Data Lives in Files** [] Like Episode, WAFL stores meta-data in files. WAFL's three meta-data files are the inode file, which contains the inodes for the file system, the block-map file, which identifies free blocks, and the inode-map file, which identifies free inodes. The term 'map' is used instead of 'bit map' because these files use more than one bit for each entry. The block-map file's format is described in detail below." EX1015 ("Hitz") at p. 8.

113. Indeed, files can transparently map to multiple data and meta-data blocks in WAFL. When a file is modified, any of its blocks that need to change are copied first, using standard CoW techniques. "Figure 2 … shows that files are made up of individual blocks and that large files have additional layers of indirection between the inode and the actual data blocks. In order for WAFL to boot, it must be able to find the root of this tree, so the one exception to WAFL's write-anywhere rule is that the block containing the root inode must live at a fixed location on disk where WAFL can find it." EX1015 ("Hitz") at pp. 9-10, Figure 2 (reproduced below).

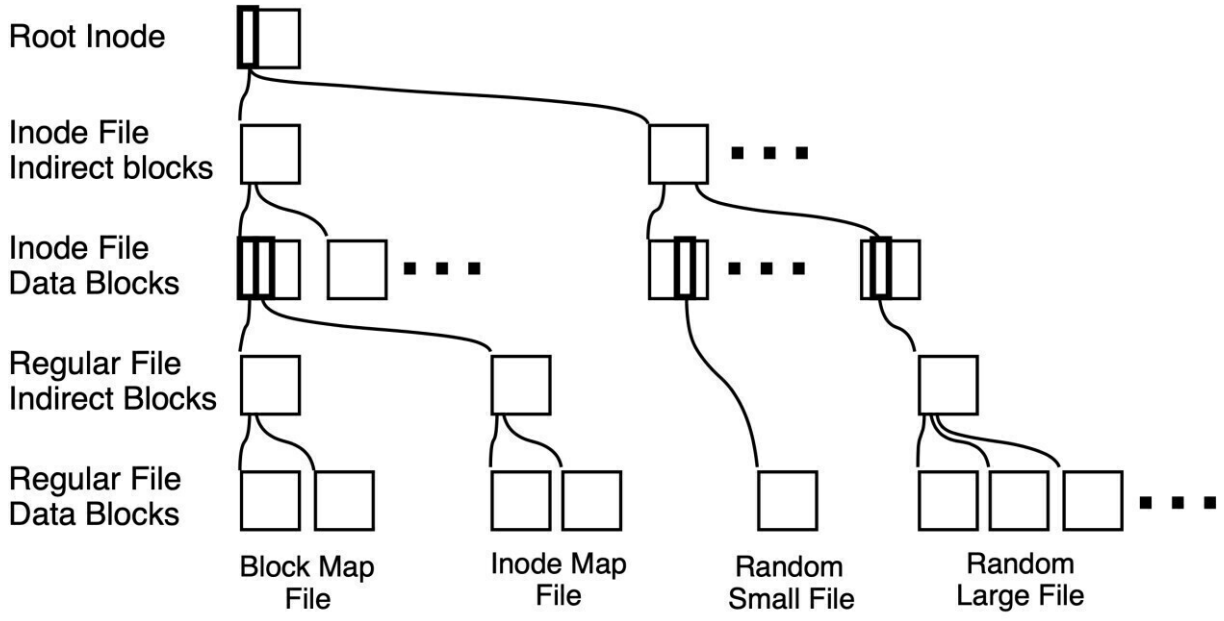


Figure 2
A more detailed view of WAFL's tree of blocks.

114. Netapp details how snapshots work in the presence of file modifications. "**3.4. Snapshots** … Understanding that the WAFL file system is a tree of blocks rooted by the root inode is the key to understanding Snapshots. To create a virtual copy of this tree of blocks, WAFL simply duplicates the root inode.

Figure 3 shows how this works. Figure 3(a) is a simplified diagram of the file system in Figure 2 … Figure 3(b) shows how WAFL creates a new Snapshot by making a duplicate copy of the root inode. … Figure 3(c) shows what happens when a user modifies data block D. WAFL writes the new data to block D' on disk, and changes the active file system to point to the new block. The Snapshot still references the original block D which is unmodified on disk. Over time, as files in the active file system are modified or deleted, the Snapshot references more and more blocks that are no longer used in the active file system. The rate at which files change determines how long Snapshots can be kept on line before they consume an unacceptable amount of disk space." EX1015 ("Hitz") at pp. 10-11, Figure 3 (reproduced below).

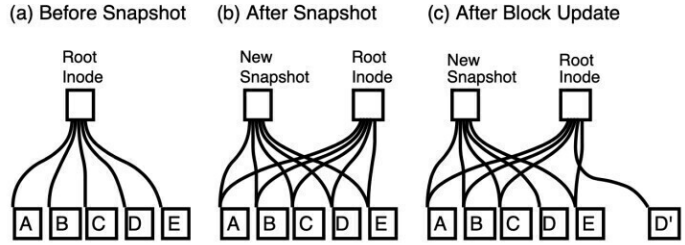


Figure 3
WAFL creates a Snapshot by duplicating the root inode that describes the inode file. WAFL avoids changing blocks in a Snapshot by writing new data to new locations on disk.

115. Initially, Netapp's snapshots were all read-only, with modifications allowed only at the top of the snapshot stack. This created a linear set of several read-only snapshots followed by the primary writeable storage view, further visible to applications as a file system. Netapp's next evolution was to permit one to create writeable snapshots, by utilizing CoW of a read-only snapshot.

116. Netapp's developed a "system and method for creating a writable clone of a read- only volume. … [A] base snapshot is generated on a source volume on a source storage system and is duplicated as a read-only base snapshot replica on a target volume on a destination storage system. A copy ('clone') is then substantially instantaneously created from the read-only base snapshot replica, thereby creating a writable clone of a read-only volume." EX1016 ("Fair") at 6:18-41.

117. With writeable clones enabled, Netapp filers could now create a (non-linear) tree of snapshots, some read-only and some cloned and writeable. "[M]ultiple clones 1350 may be spawned from the same replica base snapshot 1320; in this case, information is recorded by setting the same snapshot bit in bit fields in both clone's storage label 30 files 1390." EX1016 ("Fair") at 21:27-31.

118. Now that Netapp had both read-only and writeable snapshots, the next evolution was to transmit a snapshot's data from one networked filer appliance to another, thus creating an identical mirror of the source filer.

119. "Computerized data has become critical to the survival of an enterprise. Companies must have a strategy for recovering their data should a disaster such as a fire destroy the primary data center." EX1017 ("Patterson I") at p. 2.

120. "Mirrors keep backup data on-line and fully synchronized with the primary store, but they do so at a high cost in performance (write latency) and network bandwidth." EX1017 ("Patterson I") at p. 2.

121. "SnapMirror [is] a technology which implements asynchronous mirrors on Network Appliance filers. SnapMirror periodically transfers self-consistent snapshots of the data from a source volume to the destination volume. The mirror is on-line, so disaster recovery can be instantaneous. Users set the update frequency. If the update frequency is high, the mirror will be nearly current with the source and very little data will be lost when disaster strikes. But, by lowering the update frequency, data managers can reduce the performance and network cost of maintaining the mirror at the risk of increased data loss." EX1017 ("Patterson I") at pp. 2-3.

122. Like any operating system and appliance, Netapp utilizes caching heavily. *See also* §VII.D above.

123. "During a write episode, WAFL allocates disk space for all the dirty data in the cache and schedules the required disk I/O." EX1013 ("Hitz") at p. 12. "A file system with an NVRAM write cache does all the same steps, except that it copies modified data into NVRAM instead of waiting for the data to reach disk." EX1015 ("Hitz") at p. 13. "WAFL's technique for keeping Snapshot data self consistent is to mark all the dirty data in the cache as 'IN_SNAPSHOT.'" EX1015 ("Hitz") at p. 17.

124. "WAFL caches inode data in two places: in a special cache of in-core inodes, and in disk buffers belonging to the inode file. When it finishes write

allocating a file, WAFL copies the newly updated inode information from the inode cache into the appropriate inode file disk buffer, and clears the IN_SNAPSHOT bit on the in-core inode." EX1015 ("Hitz") at p. 17.

125. The requested block is then retrieved from disk and stored in a buffer cache of the memory as part of a buffer tree of the file. The buffer tree is an internal representation of blocks for a file stored in the buffer cache and maintained by the file system." EX1016 ("Fair") at 3:1-5.

126. "[P]eriodic updates use the primary volume as a giant write cache and it has long been known that write caches can reduce I/O traffic." EX1017 ("Patterson I") at p. 8.

127. "[W]e used two Network Appliance F760 filers directly connected via Intel GbE. Each utilized an Alpha 21164 processor running at 600 MHz, with 1024 MB of RAM plus 32 MB non-volatile write cache." EX1017 ("Patterson I") at p. 12.

128. Any storage system with snapshots would include at least one complete "base" or "root" and one or more incremental (leaf) images, called "snapshots." Because snapshots naturally contain incomplete information, they alone cannot be mounted as a file system. Any blocks not included in one snapshot must be located in a previous point in time snapshot; and those not found there have to be located in the next previous snapshot; and so on until one gets to the base/root image which is

by definition complete. Thus, the process of traversing several snapshots all the way down to the root image would result in merging or unifying one or more disparate snapshots with the root/base image, to produce a merged or unified view of the file system that is consistent and cohesive.

129. EX1018 (Patterson II) describes a system for "[r]esynchronization of a target volume with a source volume" in the presence of snapshots. EX1018 (Patterson II), Title. This is useful to mirror two or more volumes in any direction.

130. Patterson II describes "[a]n improved method and apparatus for quickly and efficiently updating the original source volume and original target volumes after the original source volume has become temporarily unavailable. The original target volume is characterized as a source volume while the original source volume is temporarily unavailable. Transfer lists of different data blocks are generated. Data blocks not originally found on a source are copied to the target. Data blocks included on a target that were not found on the source are removed. By focusing upon specific data blocks, this technique avoids the use of filer overhead and other computational resources that would be expended if the entire volume were recopied." EX1018 (Patterson II), Abstract.

131. Patterson II's Figure 2 (reproduced below) describes this process.

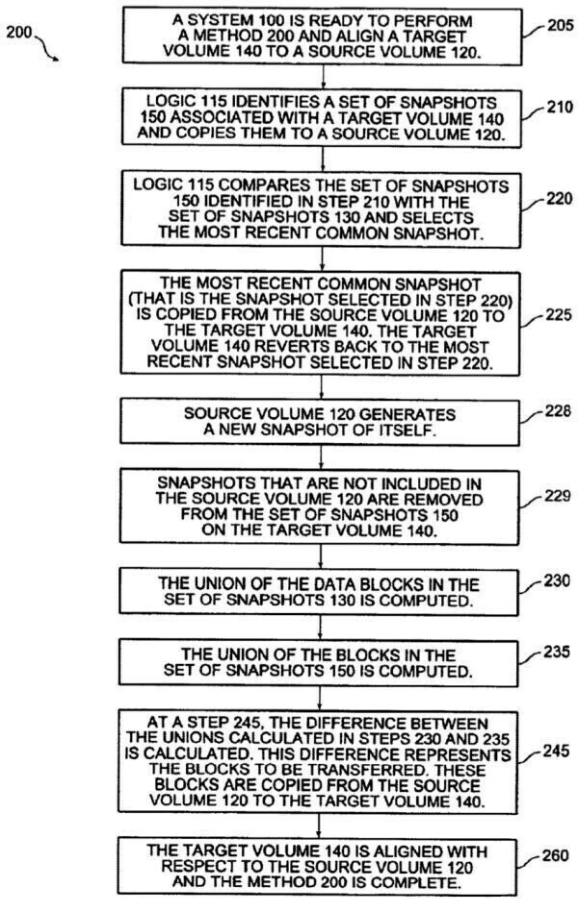


FIG. 2

132. In Figure 2, the system starts by comparing snapshots at the source and destination volumes, to identify a set of differences that have to be transmitted (steps 205-229). The next three steps actually calculate the merged union of the snapshots that need to be transferred, and then transferring those unions.

133. "At a step 230, the union of the data blocks in the set of snapshots 130 is computed. This union will preferably include available and allocated data blocks from any one or more target snapshots 130. This step is preferably performed by the source volume 120. Steps 235 and 245 occur simultaneously with step 230." EX1018 (Patterson II), 5:6-10.

134. "At a step 235, the union of the blocks in the set of snapshots 150 is computed. This union will preferably include available and allocated data blocks from any one or more target snapshots 150. This step is preferably performed by the source volume 120. Step 235 is performed at the same time as steps 230 and 245." EX1018 (Patterson II) 5:11-16.

135. "At a step 245, difference between the unions calculated in steps 230 and 235 is calculated. This difference represents the blocks to be transferred. These blocks are copied from the source volume 120 to the target volume 140. In this way, the target volume 140 becomes synchronized with respect to blocks present on the source volume 120." EX1018 (Patterson II) at 5:17-22.

136. Finally, at step 260, the synchronization of the two volumes is complete. A POSITA would easily recognize that the steps Patterson II disclose merge/unify data blocks from multiple snapshots into a set to transmit. This unioning takes place at the block level, because Patterson II's snapshots are performed at the block level (the file system resides logically above). EX1018 (Patterson II), Figure 1 (reproduced below).

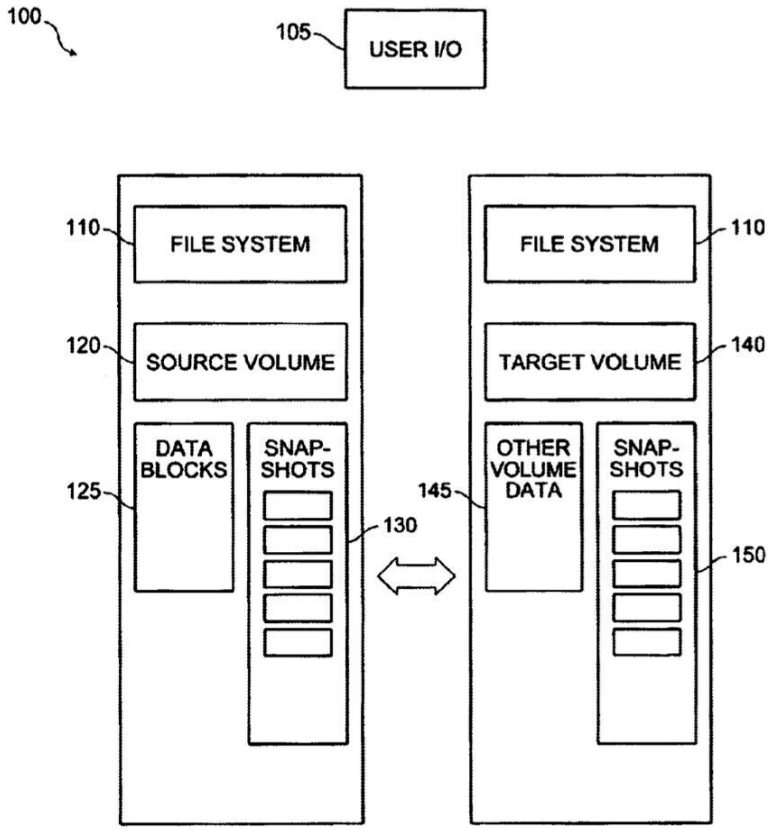


FIG. 1

137. In Patterson II, the software component that performs said unification of snapshots happens to be driven by a file system. "File system software synchronizes the target volume with the source volume. First, the file system software removes snapshots from a target volume if the snapshots are not included in the source volume's snapshot list. Second, the file system software adds the set of data blocks identified above (that is the set of data blocks that are included in the source volume and not included in the target volume) to its memory. Lastly, the file system software adds snapshots to the target volume if the snapshots are included in the source volume's snapshot list and not in the target volume's snapshot list. At

this point, the target volume includes the data blocks that are present on the source volume." EX1018 (Patterson II) at 2:22-34.

138. Lastly, in Patterson II, "sources and volumes can be synchronized dynamically, using a WAFL (Write Anywhere File Layout) system using RAID (Redundant Arrays of Independent Disks) architecture. However, various other types of file systems involving redundant copies of data can also be used." EX1018 (Patterson II) at 2:47-53. A POSITA would know that RAID is software that operates at the block level. *See* §VII.B above.

F. Indexing

139. Indexing is a general computing technique to organize a catalog for finding information more efficiently (just as a library contains a catalog of all books it has and their shelf location).

140. An "indexed search" is "[a] search for an item of data that uses an index to reduce the amount of time required." EX1013 (Microsoft Computer Dictionary), p. 247.

141. Similarly, users are familiar with many forms of index searches, such as on the WWW. A "Web index" is "[a] Web site intended to enable a user to locate other resources on the Web. The Web index may include a search facility or may merely contain individual hyperlinks to the resources indexed." EX1013 (Microsoft Computer Dictionary), p. 505.

142. There are several forms of indexes and data structures possible. An "index" (definition 1) is "1. A listing of keywords and associated data that point to the location of more comprehensive information, such as files and records on a disk or record keys in a database. 2. In programming, a scalar value that allows direct access into a multi-element data structure such as an array without the need for a sequential search through the collection of elements. *See also* array, element (definition 1), hash, list." EX1013 (Microsoft Computer Dictionary), p. 247.

143. Indexes can be created and managed in various forms, including storage systems, file systems, databases, and more. To "index" (definition 2) is "1. In data storage and retrieval, to create and use a list or table that contains reference information pointing to stored data. 2. In a database, to find data by using keys such as words or field names to locate records. 3. In indexed file storage, to find files stored on disk by using an index of file locations (addresses). 4. In programming and information processing, to locate information stored in a table by adding an offset amount, called the index, to the base address of the table." EX1013 (Microsoft Computer Dictionary), p. 247.

G. Virtualization

144. Among computer scientists, there is an axiom stating that "you can solve almost any problem by adding another level of indirection." Indeed, virtualization is an indirection technique that makes one thing seem like another;

virtualization often presents a physical-world artifact as one that appears as an illusion or "fake." For something to be "virtual", it is "[o]f or pertaining to a device, service, or sensory input that is perceived to be what it is not in actuality, usually as more 'real' or concrete than it actually is." EX1013 (Microsoft Computer Dictionary), p. 497.

145. Indeed, the Microsoft Computer Dictionary includes 35 examples and definitions of virtualization. EX1013 (Microsoft Computer Dictionary), pp. 497-500.

146. In one example, a "virtual machine" is "[s]oftware that mimics the performance of a hardware device, such as a program that allows applications written for an Intel processor to be run on a Motorola chip. *Acronym*: VM." EX1013 (Microsoft Computer Dictionary), p. 498.

147. In another example, "virtual memory" is "[m]emory that appears to an application to be larger and more uniform than it is. Virtual memory may be partially simulated by secondary storage such as a hard disk. Applications access memory through virtual addresses, are translated (mapped) by special hardware and software onto physical addresses. *Acronym*: VM. *Also called* disk memory. *See also* paging, segmentation." EX1013 (Microsoft Computer Dictionary), p. 498.

148. In yet another example, "virtual reality" is "[a] simulated 3-D environment that a user can experience and manipulate as if it were physical. The

user sees the environment on display screens, possibly mounted in a special pair of goggles. Special input devices, such as gloves or suits fitted with motion sensors, detect the user's actions. *Acronym*: VR.". EX1013 (Microsoft Computer Dictionary), p. 499.

149. Virtualization has been applied to storage systems, disk devices, and drivers. A "virtual device driver" is "[s]oftware in Windows 95 that manages a hardware or software system resource. If a resource retains information from one access to the next that affects the way it behaves when accessed (for example, a disk controller with its status information and buffers), a virtual device driver must exist for it." EX1013 (Microsoft Computer Dictionary), p. 498.

150. A "virtual disk" or "RAM disk" is "[s]hort for random access memory disk. A simulated disk drive whose data is actually stored in RAM memory. A special program allows the operating system to read from and write to the simulated device as if it were a disk drive. RAM disks are extremely fast, but they require that system memory be given up for their use. Also, RAM disks usually use volatile memory, so the data stored on them disappears when power is turned off. Many portables offer RAM disks that use battery-backed CMOS RAM to avoid this problem. *See also* CMOS RAM. *Compare* disk cache." EX1013 (Microsoft Computer Dictionary), pp. 395-396, 498.

151. Virtualization is sometimes said to be "logical" representation of physical reality. "Logical" is "[o]f or pertaining to a conceptual piece of equipment or frame of reference, regardless of how it may be realized physically. *Compare* physical." EX1013 (Microsoft Computer Dictionary), p. 288 ("logical" definition 2).

152. A "logical drive" or "logical device" is "[a] device named by the logic of a software system, regardless of its physical relationship to the system. For example, a single floppy disk drive can simultaneously be, to the MS-DOS operating system, both logical drive A and drive B." EX1013 (Microsoft Computer Dictionary), p. 288.

153. Moreover, one can virtualize files and file systems. This was the subject of much of my graduate-level research and became part of my PhD Dissertation. *See* §II above and EX1004 (Zadok Curriculum Vitae).

154. Virtualization, of course, has been applied to networking as well. A "virtual network" is "[a] part of a network that appears to a user to be a network of its own. For example, an Internet service provider can set up multiple domains on a single HTTP server so that each one can be addressed with its company's registered domain name. *See also* domain name, HTTP server (definition 1), ISP." EX1013 (Microsoft Computer Dictionary), p. 499.

155. A "virtual LAN" is "[s]hort for **virtual l**ocal **a**rea **n**etwork. A local area network consisting of groups of hosts that are on physically different segments but that communicate as though they were on the same wire. *See also* LAN." EX1011 (Microsoft Computer Dictionary), p. 498.

156. One of the most famous uses of a (secure) virtual network is a private one. A "virtual private network" ("VPN") is "[a] set of nodes on a public network such as the Internet that communicate among themselves using encryption technology so that their messages are as safe from being intercepted and understood by unauthorized users as if the nodes were connected by private lines. 2. A wide area network formed of permanent virtual circuits (PVCs) on another network, especially a network using technologies such as ATM or frame relay. *Acronym*: VPN. *See also* ATM (definition 1), frame relay, PVC." EX1011 (Microsoft Computer Dictionary), p. 499. A POSITA would have understood definition 2 of VPN to be a form of an overlay network because it is a collection of individual virtual network segments.

VIII. THE '844 PATENT

A. Overview

157. The '844 patent describes "technology for, among other things root image caching and indexing for block-level distributed application management." EX1001 ('844 patent), Abstract. This "involves storing blocks of a root image on

a first storage unit and storing blocks of leaf images on respective second storage units [which] … include additional data blocks not previously contained in the root image and changes made by respective compute nodes to the blocks of the root image." *Id*. A POSITA would have recognized that these leaf images correspond to storage snapshots as I describe in §VII.E above.

158. The '844 patent then describes common "caching blocks of the root image that have been accessed by at least one compute node." *Id*. Caching was well known and ubiquitous as I describe in §VII.D above.

159. Finally, the '844 patent describes generating and "receiving indexing results pertaining to the root image from one compute node and providing the results for other compute nodes." *Id*. Again, indexing of data and files was well known in the art, as I described in §VII.F above.

160. The '844 patent describes a background where, "as the internet has expanded and computers have multiplied, the need for clustered computing such as High Performance Computing (HPC) has increased." and that "[c]lustered computing involves multiple compute nodes, usually a server grid, that work together to achieve a common task." *Id.*, 1:31-44.

161. The '844 patent describes the trend towards global file system management: "[i]n large-scale systems such as this, a trend in software deployment

is to centralize data management on a globally accessible file system with stateless computing nodes." *Id*., 1:38-40.

162. That led to the need to manage individual nodes' boot software and/or boot images. "A common example of this is Operating System (OS) software image management, where the compute nodes are activated with the distributed application environment by either diskless booting protocols or remote software installation to local storage. Under this architecture, a boot image is required for each compute node in the cluster." *Id*., 1:40-46.

163. OS boot images contain the OS kernel, libraries, applications, and more. "The boot image necessarily contains the kernel; it may additionally contain the application software that is intended to be run on the compute node." *Id*., 1:46-48.

164. The '844 patent explains a concern with managing so many boot images, many of which may be (nearly or) identical. "The primary concern in clustered computing is low cluster bring-up time. The software that provides the boot images for the cluster typically stores a master boot image. It may then either pre-create clones of this master image for each such server, or it may create them 'on the fly.'" *Id.*, 1:49-67.

165. Thus, "[c]reating a boot image on the fly involves copying the entire contents of the master image, which are typically in the range of 5-15 GB." *Id*., 1:54-

56. This can be slow. "Even with a significant amount of bandwidth by today's standards, this method will result in a large bring-up time." *Id*., 1:56-58.

166. One proposed solution is that "Pre-creating a boot image for each server is advantageous from the point of view of cluster bring-up time." *Id*., 1:59-60. But such a solution could waste disk/storage space "since one often does not know in advance how many servers will ever be booted, this scheme may result in wasted disk space." *Id*., 1:60-63. Thus, "both [methods] suffer from the same major problem—updating the boot image(s) for the cluster is cumbersome, as it means updating a number of copies of the boot image." *Id.*, 1:64-67.

167. The '844 Background section then discusses how "[i]n a branching store file system, a read-only base image (or 'root' image) of the application environment is created." *Id.*, 2:13-24.

168. "The root image is accessible by all compute nodes in the cluster" and "[c]hanges made by a compute node to the root image are stored in a 'leaf' image unique to that compute node." (*i.e.*, a snapshot). *Id*., 2:15-18.

169. To make this work, a software driver, component, or filter "operates between the compute nodes and the file system(s), which merges the changes recorded on the leaf images with the root image and delivers the result to the appropriate compute node." *Id*., 2:18-21.

170. The benefit is that, "[f]rom the point of view of the compute node, it is running its own unique and cohesive instance of the application environment." *Id*., 2:22-24.

171. Nevertheless, "[w]hile this system allows for creation of boot images on the fly without severely diminishing bring-up time, a separate version of the system must be created for each unique operating system because data is stored at the file system level (i.e., on a 'per file basis')." *Id*., 2:24-28.

172. Therefore, the '844 patent argues that "migrating a computing cluster from one operating system to another is much more complicated than simply installing a new root image containing the new OS." *Id*., 2:28-31.

173. After discussing and admitting to this known prior art, the '844 patent describes an example system that includes "a general purpose computing system environment, such as compute node 100." *Id*., 4:27-29.

174. "[C]ompute node 100 typically includes at least one processing unit 102 and memory 104" (e.g., RAM, ROM, and/or flash memory). *Id*., 4:29-36.

175. The system may also have additional common features/functionality, such additional removable 108 and/or non-removable 110 storage (*e.g.*, disks, tapes), communications connections 112, input devices (*e.g.*, keyboard) 114, and output devices (*e.g.*, monitor) 116. *Id*., 4:36-5:10. *See also* §VII.A above.

176. Memory 104, removable storage 108, and non-removable storage 110 are all examples of computer storage media" which "includes, but is not limited to, RAM, ROM, flash memory, DVDs, …, magnetic storage devices, or "any other medium which can be used to store the desired information and which can be accessed by compute node 100." *Id*., 4:46-55, Fig. 1 (reproduced below).

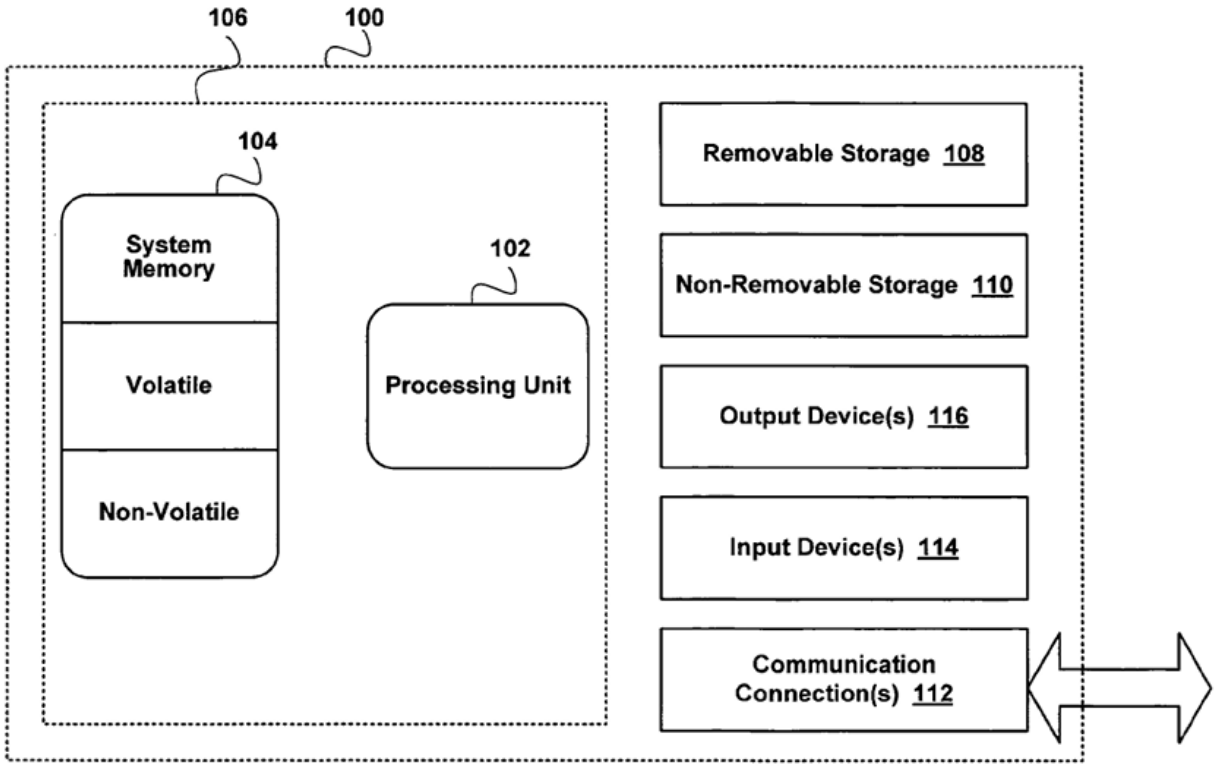


FIG. 1

'844 Patent, Figure 1

177. The '844 patent then describes components of the system. "FIG. 2 is a diagram of a system 200 for root image caching and indexing in a block-level distributed application environment, in accordance with various embodiments of the

present disclosure. In one embodiment, system 200 is implemented in a multi-computer system, such as an HPC cluster. In one embodiment, the application environment includes an operating system. In other embodiments, the application environment may contain other applications." *Id.*, 5:12-26, Fig. 2 (reproduced below).

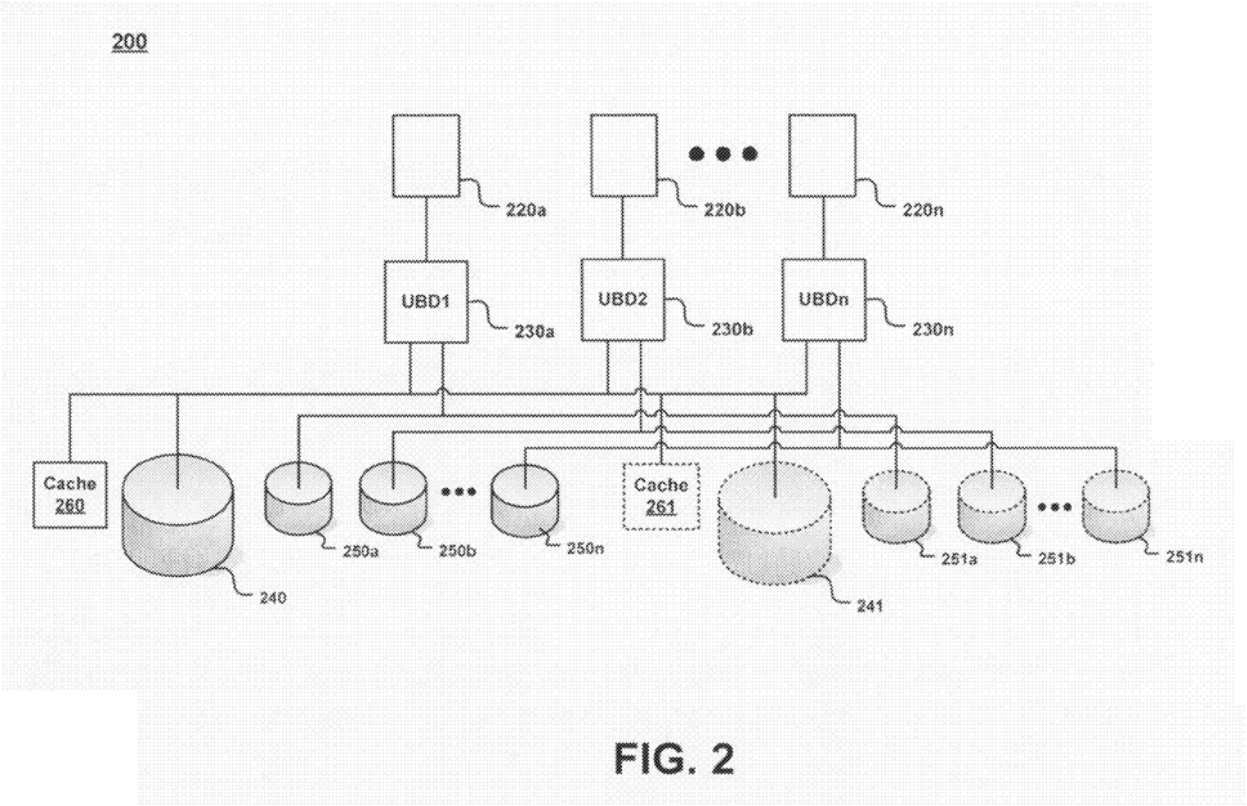


FIG. 2

'844 Patent, Figure 2

178. It then describes how the nodes and the storage they use. "System 200 has a number of compute nodes 220*a-n* coupled to first storage unit 240 and a corresponding second storage unit 250*a-n* though a corresponding union block device (UBD) 230*a-n*. To the compute nodes 220*a- n*, it appears that they have

access to their own version of a distributed application environment. However, a separate and complete boot image is not created and stored for each compute node 220*a-n*." *Id*. Of note is the disclosed "UBD", which I describe in more detail in §VIII.C below.

179. Instead of having separate boot images for each node, "a first storage unit 240" "store[s] blocks of a root image of an application environment. The root image contains data initially common to the compute nodes 220*a-n*" and "is not changed by the compute nodes 220*a-n*. *Id*., 5:27-32.

180. In one embodiment, compute nodes 220*a-n* have read-only access to the first storage unit 240. Each compute node 220*a-n* also has a corresponding second storage unit 250*a- n* for storing a leaf image, which in the challenged claims includes "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image" or similar language. *Id*., 5:33-34, 10:61-65, 11:32-35, 12:4-8, 12:47-50.

181. In one embodiment, "each compute node 220*a-n* has a corresponding second storage unit 250*a-n* for storing a leaf image. The first storage unit 240 and second storage units 250*a-n* may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof." *Id.*, 5:33-58.

182. The leaf image may contain blocks of new data, blocks of changed data, or other blocks of data unique to the individual compute node." *Id*. Thus, "a leaf image will describe the changes made by the respective compute node 220*a-n* to its instance of the application environment." *Id*.

183. One main reason why the challenged claims were allowed is the addition of "only" in "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image" or similar language. *Id*., 5:33-34, 10:61-65, 11:32-35, 12:4-8, 12:47-50.

184. The '844 patent states that "[w]ith respect to changes to the root image, only the specific blocks that are changed are stored in the leaf image. … One compute node (e.g., compute node 220*a*) desires to make a change to this file which involves a modification of only a few specific blocks of the file (e.g., blocks 4-9). In this example, only the modified blocks (e.g., blocks 4-9) will be stored in the compute node's leaf image (e.g., leaf image stored on second storage device 250*a*) plus some small overhead." *Id.*, 5:33-58.

185. This claim language differs from the specification, which states that the leaf image "includes, but is not limited to, new data blocks for the compute node and blocks of the root image that the compute node has changed." *Id*., 7:64-66.

186. The first storage unit 240 and second storage units 250*a-n* may each be contained on separate physical storage devices, on separate logical spaces on the same storage device, or any combination thereof. *Id*., 5:34-38.

187. I note that Claim 23 requires that the "leaf image portion" comprise "only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image." *Id*., 13:8-12. A POSITA would have understood this to claim snapshots as I describe in §VII.E above.

188. Next, the '844 patent describes a "union block device" (UBD) that logically merges blocks between a root image and one or more lead images, just as snapshotting systems do. I describe this UBD and its relationship to commonly known snapshots in detail in §VIII.C below.

189. The '844 patent describes an embodiment that uses caching to improve performance. "[T]he present disclosure provides for caching of portions of the root image. Because several compute nodes in a cluster may often access the same data (e.g., same drivers, same library files, etc.) on the root image, tremendous speed improvements can be realized by caching such data in cache 260." *Id.*, 6:38-49. A POSITA would have known that caching can be added to many systems to improve performance and has been in operation for many years, as I detail in §VII.D above.

190. Moreover, the '844 patent describes an embodiment where "the cache may be contained within a single storage appliance, along with the first storage unit 240 and the second storage units 250*a-n*." *Id.*, 6:38-49. A POSITA would have known that sharing any data (volatile or non-volatile caches or otherwise) was also well known in the art. *See* §VII.C above.

191. Finally, the '844 patent describes an embodiment "for indexing the root image" because "it is beneficial for each compute node to have access to an index of its file system." *Id.*, 7:18-25. A POSITA would have known that indexing of any data was also well known in the art, as I describe in §VII.F above. The '844 patent further describes how indexing can be shared with multiple nodes: again a POSITA would have known how to share data across the network. *See* §VII.C above. I discuss the '844 patent's indexing disclosures in more detail in §VIII.B below.

B. The '844 Patent Discloses Indexing at the Block Layer

192. The '844 patent claims "indexing" but does not provide sufficient details as to *how* the indexing is performed. A POSITA would have known that there are multiple forms in which indexes can be organized. *See* §VII.F above.

193. The '844 patent further discloses that that indexes can be shared with others. "It is appreciated that the indexing results can be provided to the other compute nodes in a number of ways. For example, the results may be stored on a shared storage unit (such as, but not limited to, cache 260), the results may 40

provided directly to the other compute nodes, etc." EX1001 ('844 patent) at 7:37-41. I note that the '844 patent does not say how such index data can be shared with others. A POSITA, however, would have known that data can be easily shared across the network using block-layer (*e.g.*, iSCSI) or file-layer protocols (*e.g.*, NFS). *See* §VII.B.1 above.

194. For any program to be able to index files at the file system layer, it would require access to a fully mountable, complete file system. As described above (*see* §VII.B above), this requires access to a base image plus one or more snapshots that provide a complete view of the file system at a point. A POSITA would have also understood that merely taking some subset of blocks (*e.g.*, an incremental snapshot) does not provide a valid format file system; an incremental "leaf" snapshot alone cannot even be mounted or accessible as a file system because critical data structures that make up files, folders, and names are missing.

195. The '844 patent describes that indexing takes place at the logical block layer, not at the file system layer, for four reasons.

196. First, the '844 patent states that file systems are "agnostic" to the fact that their lower-level blocks and sectors are spread around a root image plus one or more leaf images (*i.e.*, snapshots). "Step 520 involves the compute node indexing the root image portion of its file system. In such a case, the compute node is to some extent agnostic to the fact that its file system is divided between a root image portion

and a leaf image portion." *Id.*, 9:48-52. A POSITA would have understood this passage to mean that indexing takes place at either the root image or one of the leaf images.

197. Second, the '844 patent clearly states that it indexes only the leaf image (an incremental snapshot) independently from file systems. "Further down the road, the benefits of indexing the root image separately from the leaf image are realized when the compute node requires that its file system be re-indexed. At step 560, the compute node re-indexes its file system by re-indexing its corresponding leaf image portion with the previous indexing results of the root portion. Thus, by operating at the block level, embodiments of the present disclosure provide file system and operating system independent systems and methods for distributing an application environment to a compute node." *Id.*, 9:62-10:4. A POSITA would have understood that this can only take place by indexing at the block layer. Indeed, it would be impossible to index files in a leaf image (incremental snapshot) because it cannot even mount as file system volume.

198. Third, the '844 patent reinforces the purported benefit of being able to quickly index only the leaf image because it is small. "Thereafter, the compute nodes 220*a-n* need only be concerned with indexing their respective leaf image, which are relatively small compared to the root image." *Id.*, 7:31-33. This further

reinforces that the idea that the '844 patent's intended indexing is of the data blocks/sectors of the root/leaf images, which can only take place at the block layer.

199. Fourth, the '844 patent's claim 19 claims in its preamble "a method for indexing file systems" but its actual limitation not once mention a file, file name, or file system structure. Rather, it mentions "data blocks" of root and/or leaf images. A POSITA would have understood that the indexing here operates at the block/image level; that is, when the preamble states "indexing file system," a POSITA would have understood it to be nothing more than an outcome borne out of the fact that file systems a logically structured above block devices. Stated differently, every file that a user or application can access (*e.g.*, by name), eventually translates to one or more disk blocks at the block layer; thus, it is natural to discuss (indexed) blocks that relate to files and hence their file systems, even when actual indexing operations take place at the block layer. *See* §VII.B above.

C. The '844 Patent's Union Block Device (UBD)

200. I note that the '844 patent does not define what a UBD is. The patent describes that UBDs may be low-level drivers. "In one embodiment, UBDs 230*a-n* are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 220*a-n*. … Because UBDs 230*a-n* operate below the file system, they are concerned merely with the

blocks of data themselves, rather than files they form." EX1001 ('844 patent) at 5:60-6:1. Many storage drivers reside below file systems. *See* §VII.B above.

201. UBDs are said to "determine what leaf image (from the appropriate second storage unit 250) is needed for portions of the application environment that their respective compute nodes 220*a-n* have changed. UBDs 230*a-n* also locate the portions of the application environment that are not changed by their respective compute nodes 220*a-n*. These portions may reside in the root image. There may also be intermediate images (not depicted in FIG. 2) comprising versions of the root image from which a compute node's instance of the application environment is derived." *Id.*, 6:4-8. A POSITA would have understood that this is no more than common traversal of snapshots. *See* §VII.E above.

202. Moreover, "UBDs 230*a-n* may also modify the leaf image in response to their respective compute node's access to its instance of the application environment. For example, upon receiving a write request from their respective compute nodes for a sector X, the UBDs 230*a- n* will create an appropriate persistent mapping for sector X and then write sector X onto their respective second storage units 250*a-n*, where sector X can then be modified." *Id.*, 6:20-27. Again, a POSITA would have understood that the UBD functionality disclosed here is nothing more than typical copy-on-write (CoW). *See* §VII.E above.

203. Lastly, UBDs logically merge different root and leaf images, which are nothing more than snapshots. It is also natural to extend them to integrate data stored in a cache, which is yet another sort of snapshot in that it contains a subset of the data from the backing store (*see* §VII.D above). Thus "Step 330 involves merging the blocks of the root image, the blocks of the leaf image, and the relevant blocks of the cache, if any, to create the application environment. In other words, the merging occurs at an operational level between the file system of a compute node and the first storage unit, the corresponding second storage unit, and the cache. Once the application environment has been created, it will appear to the compute node as one cohesive image rather than a base image plus its additions, deletions, and modifications. To the compute node, it appears that it has access to its own unique version of an application environment. However, a separate and complete boot image is not actually stored for the compute node." *Id.*, 8:13-25. A POSITA would have recognized this merging functionality to be no different than what prior art snapshot systems already disclosed. *See* §VII.E above.

D. Prosecution History

1. The '477 Application

204. U.S. Application No. 11/709,477 ("the 477 Application"), which matured into the '844 patent, was a continuation-in-part of U.S. Patent Application

No. 11/395,816 ("the 816 Application"), which was a continuation-in-part of U.S. Patent Application No. 11/026,622 ("the 622 Application").

205. The '477 Application contained five independent claims (claims 1, 7, 14, 19, and 23). The first four stated that the leaf images comprised "additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image." The fifth (claim 23) stated that the leaf images comprised "additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image." EX1002 ('844 File History) at 25-30.

206. The PTO issued a First Office Action on January 5, 2010. The Examiner rejected claims 1-13 as obvious over U.S. Patent Application Publication No. 2003/0126242 A1 ("Chang") in view of U.S. Patent No. 6,101,576 ("Kobayashi"). EX1002 (File History for U.S. Patent No. 8,332,844) at, 59-64. Kobayashi was only cited for teaching physical caches, and the Examiner found it would have been obvious to a POSITA at the time of the alleged invention to have combined the "detailed image/data cache" teachings of Kobayashi with the "server/client image [*sic*] assess and remotely stored boot/OS system" teachings of Chang "for the benefit of increasing the speed of data transfers from the host to client in data streaming or transfer operations as frequently accessed data is kept in high

speed memory and not constantly retrieved from slower disk sources." *Id*. Claims 14-27 (*i.e.*, all remaining claims) were rejected as anticipated by Chang. *Id*., 53-58.

207. The Applicants submitted an Amendment and Response on April 5, 2010. Therein, the first four independent claims (claims 1, 7, 14, and 19) were amended to include a new "wherein" limitation (immediately following the "comprising" clause discussed above), stating that the "leaf images [of/for] respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes" or similar language. For example, claim 7 was amended as follows:

> 7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:
>
> storing blocks of a root image of said compute nodes on a first storage unit;
>
> storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
>
> caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

208. Claim 23 was also amended to include a new "wherein" limitation (also following the above-discussed "comprising" clause), stating that the "leaf image portion does not include blocks of said root image that are unchanged by said first compute node." *Id.*, 73-78.

209. In the accompanying remarks, the Applicants stated the addition of these new "wherein" clauses overcame the Examiner's January 5, 2010 anticipation and obviousness rejections given one or more purported gaps the teachings of Chang. *Id*., 82-88. The Applicants never contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Chang. Nor did the Applicants ever expressly raise any of the dependent claim limitations as a grounds for allowance. *Id*.

210. The PTO issued a Second Office Action on May 25, 2010. The Examiner again rejected all 27 pending claims, largely citing the same anticipation and obviousness grounds (but adding citations to Chang addressing the amended "wherein" language). *Id.*, 102-125.

211. The Applicants submitted an Amendment and Response on August 31, 2010. Therein, the first four independent claims (claims 1, 7, 14, and 19) were again amended, this time to include the term "only" immediately before the "comprising" clause discussed above, so as to now state that the claimed leaf images comprise "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of [said/the] root image…" For example, claim 7 was amended as follows:

> 7. (Currently amended) A method for providing data to a plurality of compute nodes, comprising:
>
> storing blocks of a root image of said compute nodes on a first storage unit;
>
> storing leaf images for respective compute nodes on respective second storage units, said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
>
> caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

212. Claim 23 was also amended (for a second time) to now state that the claimed leaf images comprise "only additional data blocks not previously contained in said root image portion and changes made by said first compute node to the blocks of said root image…" *Id.*, 166-71.

213. In the accompanying remarks, the Applicants stated these August 31, 2010 amendments overcame the Examiner's May 25, 2010 anticipation and obviousness rejections because the "client image copies" in Chang (what the Examiner had reportedly identified as the leaf images) include "blocks of the root image that are unchanged by a compute node (*e.g.*, the common OS and application blocks for storing the base boot image)." *Id.*, 176-182. Again, the Applicants never contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Chang. Nor did the Applicants explicitly raise any of the dependent claim limitations as a grounds for allowance.

Id. Applicants also stated a UBD is "a driver implemented by and for a particular compute node;" that they do not need a separate processor; that "drivers serving as interfaces between operating system file systems for a computer system and associated storage devices are well known in the art;" and that a POSITA "would understand how a driver may serve as an interface between storage devices and a file system of a computer system." *Id*., 172-173.

214. The PTO issued a Third Office Action on March 12, 2012. However, the Examiner never mentioned, let alone cited, Chang. Instead, the Examiner rejected: (1) claims 1-13 as obvious over U.S. Patent No. 7,870,106 ("Nguyen") in view of Kobayashi; (2) claims 14-27 as being obvious over Nguyen; and (3) claims 19-27 as anticipated by Nguyen. *Id.*, 192-206. Again, Kobayashi was only cited for teaching physical caches, and the Examiner found it would have been obvious to a POSITA at the time of the alleged invention to have combined the "detailed image/data cache" teachings of Kobayashi with the "server/client image assess and remotely stored boot/OS system" teachings of Nguyen "for the benefit of increasing the speed of data transfers from the host to client in data streaming or transfer operations as frequently accessed data is kept in high speed memory and not constantly retrieved from slower disk sources." *Id*. The Office Action concluded by stating that the Applicants' prior arguments were considered, but "are moot

because the arguments do not apply to any of the reference[s] being used in the current rejection." *Id*., 206.

215. The Applicants subsequently filed an Amendment on July 11, 2012. Therein, the Applicants overcame the Examiner's March 12, 2012 obviousness and anticipation rejections by stating that Nguyen was not prior art under 35 U.S.C. §103(a) given its alleged common ownership with the pending application. EX1002 (File History for U.S. Patent No. 8,332,844) at 224. The Applicants also amended several claims. This included the substitution of "including" for "comprising" in independent claims 1 and 7. *Id*., 239-245. Once again, the Applicants never contested the Examiner's findings with respect to the cache teachings of Kobayashi or the stated motivation to combine Kobayashi with Nguyen. Nor did they raise any of the dependent claim limitations as a reason for allowance. *Id*., 221-225.

216. The Applicants later stated, in their November 6, 2012 Comments on Statement of Reason for Allowance, that "[t]he amendments submitted on July 11, 2012, were directed to matters of form and to improve the clarity of the claims, and were not required to overcome the art of record." *Id*., 305.

217. Thus, the sole purported point of novelty for the claims of the '844 patent expressly raised during prosecution is the limitation, in each of the independent claims, that the leaf images can only comprise or include "additional data blocks not previously contained in said root image and changes made by respective compute

nodes to the blocks of [said/the] root image" (or the language in claim 23 added by the Applicants at the same time). A POSITA would have understood this additional language to claim snapshots as I describe in §VII.E above.

2. The '622 Application

218. A different Examiner was assigned to the "grandparent" '622 Application. That Examiner rejected all of the claims initially proffered by the Applicants (who differed, in part, from the Applicants in the '477 Application) in an April 13, 2009 First Office Action—as being anticipated by Chang. The Examiner thereafter repeatedly rejected all subsequently pending claims (in Office Actions dated August 24, 2009; March 15, 2010; August 13, 2010; and December 6, 2010) as obvious over Chang in view of U.S. Patent No. 6,751,658 ("Haun"). EX1012 (File History for U.S. Patent Appl. No. 11/026,622) at 50-63, 92-109, 144-167, 203-225, 259-285. Indeed, the Applicants ultimately abandoned the '622 Application with no claims ever being deemed allowable. *Id*., 294-295.

219. The Examiner assigned to the '622 Application repeatedly rejected every pending claim despite the prosecuting attorney (who was the same attorney prosecuting the '477 Application, at least for a significant portion of the time) making several amendments and arguments that resembled those offered in connection with the '477 Application (including the addition of the word "only" to several claims in a June 15, 2010 Amendment and Response). EX1012 (File

History for U.S. Patent Appl. No. 11/026,622) at 183-199. For example, independent claim 1 was amended to recite a "plurality of second file collections separate from the root file system, each comprising only modified portions of said distributed applications for respective ones of said plurality of compute nodes…" *Id*., 184.

220. Furthermore, while the '622 and '477 Applications were co-pending for an appreciable amount of time, the Examiner assigned to the '477 Application (unlike the Examiner assigned to the '622 Application) never relied on Haun. *See* EX1002 ('844 File History). In fact, the Examiner assigned to the '477 Application only mentioned Haun once (in the initial, January 5, 2010 Office Action), and then only to state that Haun (and another reference) were prior art of record and were "not relied upon but is considered pertinent to applicant's disclosure." *Id*., 64.

3. Priority Date

221. The '844 patent claims priority to the '816 Application (filed March 30, 2006), which in turn claims priority to the '622 Application (filed December 30, 2004). EX1002 (File History for U.S. Patent No. 8,332,844), EX1012 (File History for U.S. Patent Appl. No. 11/026,622), EX1019 (File History for U.S. Patent Appl. No. 11/395,816). I understand that, for purposes of the Petition only, Petitioner does not dispute that the Challenged Claims are entitled to an effective filing date of December 30, 2004. As stated above, my opinions have been guided by my

appreciation of how a POSITA would have understood the claims and the specification of the '844 patent at the time of the alleged invention, which I have been asked to initially consider as no earlier than December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1012).

IX. THE CLAIMS

222. The tables below include identifiers for the '844 patent claims and limitations, which are referenced throughout this Declaration.

No.	Claim Limitation
[1.pre]	A system for providing data to a plurality of compute nodes, the system comprising:
[1.a]	a first storage unit configured to store blocks of a root image of said compute nodes;
[1.b.i]	a plurality of second storage units configured to store leaf images of respective compute nodes,
[1.b.ii]	said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[1c]	a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.
2	The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.
3	The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

No.	Claim Limitation
4	The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.
5	The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.
6	The system as recited in claim 1 wherein said first storage unit is read-only.
[7.pre]	A method for providing data to a plurality of compute nodes, comprising:
[7.a]	storing blocks of a root image of said compute nodes on a first storage unit;
[7.b.i]	storing leaf images for respective compute nodes on respective second storage units,
[7.b.ii]	said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[7.c]	caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.
8	The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and providing said first portion of said data to said at least one of said compute nodes from said cache memory.
9	The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

No.	Claim Limitation
10	The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.
11	The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.
12	The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.
13	The method as recited in claim 7 wherein said first storage unit is read-only.
[14.pre]	A system for indexing file systems for a plurality of compute nodes, the system comprising:
[14.a]	a first storage unit configured to store blocks of a root image of said compute nodes;
[14.b.i]	a plurality of second storage units configured to store leaf images of respective compute nodes,
[14.b.ii]	said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and
[14.c]	a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes,

No.	Claim Limitation
[14.d]	wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units,
[14.e]	and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.
15	The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.
16	The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.
17	The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.
18	The system as recited in claim 14 wherein said first storage unit is read-only.
[19.pre]	A method for indexing file systems for a plurality of compute nodes, comprising:
[19.a]	storing blocks of a root image of said compute nodes on a first storage unit;
[19.b.i]	storing leaf images for respective compute nodes on respective second storage units,

No.	Claim Limitation
[19.b.ii]	said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;
[19.c]	merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;
[19.d]	receiving indexing results pertaining to said root image from one of said compute nodes; and providing said indexing results to the others of said compute nodes.
20	The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.
21	The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.
22	The method as recited in claim 19 wherein said first storage unit is read-only.
[23.pre]	A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:
[23.a]	receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion,
[23.b]	said leaf image portion comprising only additional data blocks not previously contained in said root image portion and, changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node,
[23.c]	wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;
[23.d]	index said root image portion; and provide the results of said indexing to a second compute node,

No.	Claim Limitation
[23.e]	wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.
24	The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.
25	The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.
26	The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.
27	The computer-readable storage medium of claim 23 wherein said root image portion is read-only.

X. OVERVIEW OF THE PRIOR ART

223. I first note that the prior art references analyzed in the grounds below are analogous to the ’844 patent because they are within the same field of endeavor and/or reasonably pertinent to one or more problems addressed by the ’844 patent. For example, all relate to storage systems, and all relate to one or more of snapshot management, using copy-on-write techniques, indexing techniques, and/or caching techniques.

A. Menage

224. Menage describes "[f]ile systems that are created and archived by providing a set of shared storage units and one or more templates, each template including a set of private storage units and a corresponding usage map." EX1004 (Menage), Abstract. Menage specifies that its invention operates "particularly in the context of multiple virtual private servers running on a single physical host machine." *Id.*, 3:56-58.

225. Menage explains that "implementing a virtual private server using traditional server technologies has been impossible because, rather than comprising a single, discrete process, a virtual private server must include a plurality of seemingly unrelated processes, each performing various elements of the sum total of the functionality required by a customer." According to Menage, "[a] virtual private server is an example of a 'virtual process,' which is a set of processes isolated partially or totally from other processes of the system." *Id.*, 3:59-67.

226. Menage further explains that "one of the difficulties in implementing multiple virtual private servers within a single physical host involves providing each server with a separate file system." *Id.*, 7:30-33. In Menage, a "file system is an organized accumulation of data within one or more physical storage devices, such as a hard disk drive or RAID (redundant array of inexpensive disks)," where "[t]he data is typically organized into 'files'" that "are stored within a plurality of 'storage

units' of the storage device, sometimes referred to as 'disk blocks' or 'allocation units.'" *Id.*, 7:33-40.

227. Menage notes that "providing a separate physical device for storing the file system of each virtual private server would be expensive and inefficient," and that "it would be desirable to store the file systems of multiple virtual private servers within the same physical device or comparatively small set of devices." *Id.*, 7:41-46.

228. Menage explains that a product called ServerXchange "provides multiple virtual processes, such as virtual private servers, with separate file systems," and that "[e]ach file system is stored in a linear set of equal-sized storage units within the same physical storage device." *Id.*, 7:47-53. However, the ServerXchange approach "still presents a number of difficulties." *Id.*, 7:54-55. For example, "an initial file system, including standard applications, utilities, databases, etc., must be duplicated for each virtual private server," which "results in copying large amounts of data from one disk location to another, consuming significant amounts of time and placing substantial memory and disk load on the physical server." *Id.*, 7:55-61. Moreover, "[s]uch extensive duplication also results in wasted storage space, since all or part of the resulting file systems are identical." *Id.*, 7:61-63. Thus, "what is needed is a technique for creating separate file systems for a

plurality of virtual private servers that does not require extensive copying or wasted storage space." *Id.*, 2:33-36.

229. Menage's Figure 3 "is a block diagram of a storage device including a set of shared storage units, a set of private storage units, and a usage map," and according to Menage, "the foregoing problems are solved by providing a set of shared storage units 302 and a set of private storage units 304." *Id.*, 3:31-33, FIG. 3 (reproduced below).

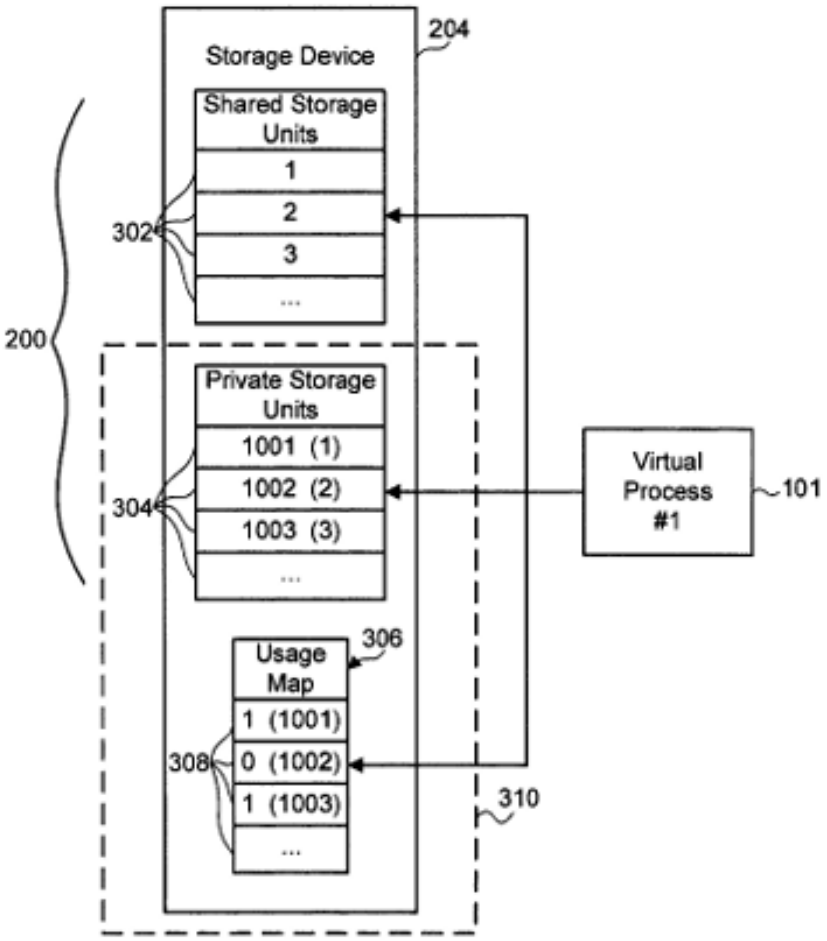


FIG. 3

Menage, Figure 3

230. In Menage, the "the set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101, such as standard application programs, utilities, databases, etc.," and "forms an initial file system 200 for each virtual process 101." *Id.*, 8:8-15.

231. "The set of private storage units 304, on the other hand, are used to store changes to the initial file system 200 that occur subsequent to the creation of the file

system 200," and "[e]ach private storage unit 304 preferably corresponds to one of the shared storage units 302." *Id.*, 8:16-19. In Figure 3, "the number in parentheses within each private storage unit 304 refers to the corresponding shared storage unit 302." *Id.*, 8:19-21.

232. Menage notes that "[w]hile the private storage units 304 are depicted as being located within the same storage device 204 as the shared storage units 302, the invention is not limited in this respect," and that in alternative embodiments, "the private storage units 304 may be located within a separate storage device 204, either within the same physical server or within a remote server." *Id.*, 8:22-27.

233. Moreover, "a usage map 306 is also provided for each set of private storage units 304" that "includes a plurality of indicators 308 for indicating whether a corresponding private storage unit 304 contains valid data." *Id.*, 8:28-31. In Menage, "the phrase 'contains valid data' means that the private storage unit 304 stores data representing a change to the initial file system 200." *Id.*, 8:32-34. In one embodiment, "the usage map 306 is embodied as a bitmap, with each indicator 308 comprising a single bit," for example, "a bit with a value of 1 may indicate that a corresponding private storage unit 304 contains valid data, whereas a bit with a value of 0 may indicate the opposite." *Id.*, 8:35-39. In Figure 3 (reproduced above), "the number in parentheses within each indicator 308 refers to the corresponding private storage unit 304." *Id.*, 8:39-42.

234. In an embodiment, "the usage map 306 is initially reset or 'initialized' to indicate that none of the private storage units 304 contain valid data," for example, in the bitmap embodiment, "all of the bits could be reset to 0." *Id.*, 8:46-49.

235. Menage notes that although "the usage map 306 is stored in the same storage device 204 as the private storage units 304," that "the usage map 306 may be stored in a different storage device 204 or in the computer memory 102," and in one implementation, "a copy of the usage map 306 is cached in the computer memory 102 or in a memory integrated with the storage device 204 to provide high-speed access to the usage map 306." *Id.*, 8:50-57.

236. In Menage, "a set of private storage units 304 for a virtual process 101 and a corresponding usage map 306 are collectively referred to as a 'template' 310." *Id.*, 8:58-61. Menage notes that although Figure 3 "illustrates only a single template 310, it should be recognized that at least one template 310 is preferably provided for each virtual process 101." *Id.*, 8:61-63. As explained with respect to Figures 4-6 of Menage, "the file system 200 of each virtual process 101 comprises a combination of the shared storage units 302 and the private storage units 304 from one or more templates 310." *Id.*, 8:63-67.

237. Menage's Figures 4-6 show a "system 400 for creating a managing the file systems 200 of a plurality of virtual processes 101." *Id.*, 9:1-4. With respect to writing a data item, as shown in Figure 4, "a system call wrapper 111, as described

with reference to FIG. 1, intercepts a system call 115 for writing a data item to a shared storage unit 202." *Id.*, 9:4-7, FIG. 4 (reproduced below).

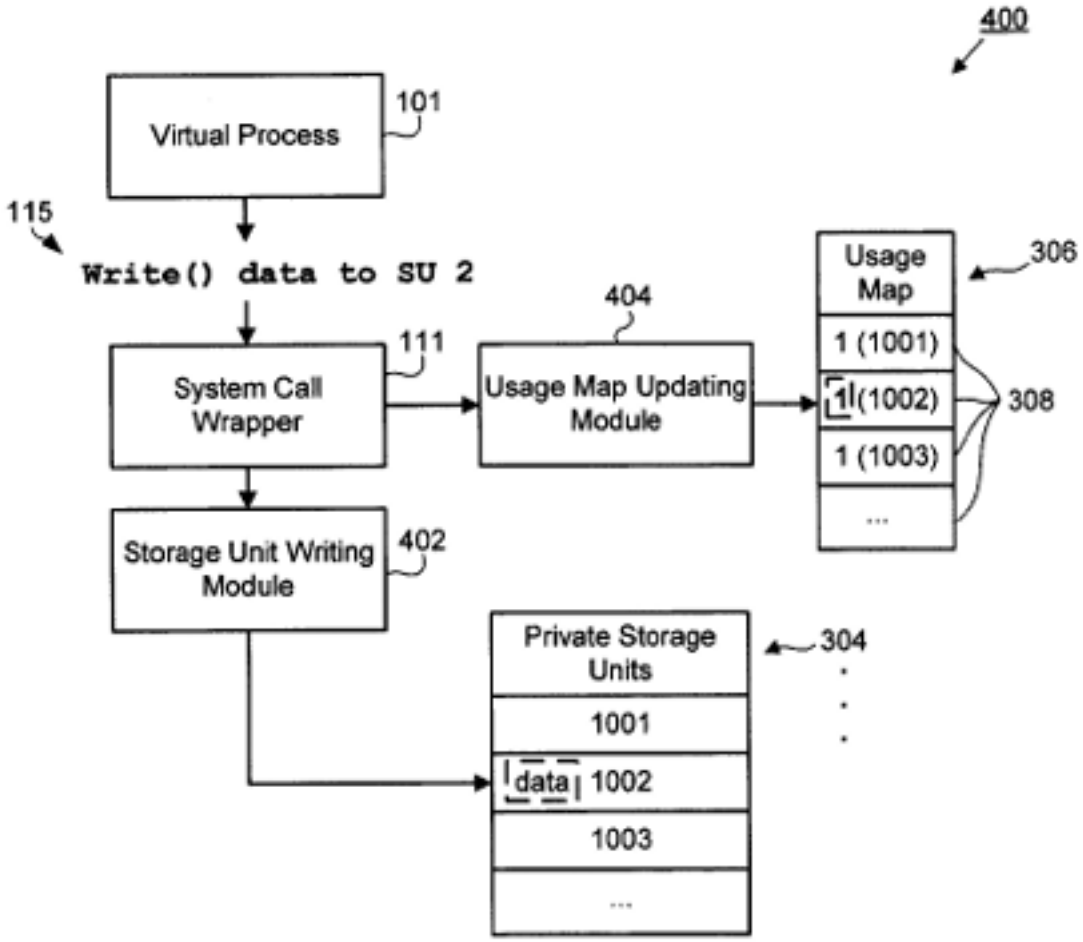


FIG. 4

Menage, Figure 4

238. As shown in Figure 4, "[r]ather than writing the data item to the shared storage unit 302, however, a storage unit writing module 402 writes the data item to the corresponding private storage unit 304." *Id.*, 9:10-12. Further, a usage map

updating module 404 stores an indication 308 in the usage map 306 that the private storage unit 304 (to which the data item is written) contains valid data." *Id.*, 9:18-21.

239. With respect to reading a data item, as shown in Figure 5, "the system call wrapper 111 is also configured, in one embodiment, to intercept a system call 115 for reading a data item from a shared storage unit 302." *Id.*, 9:29-32, FIG. 5 (reproduced below).

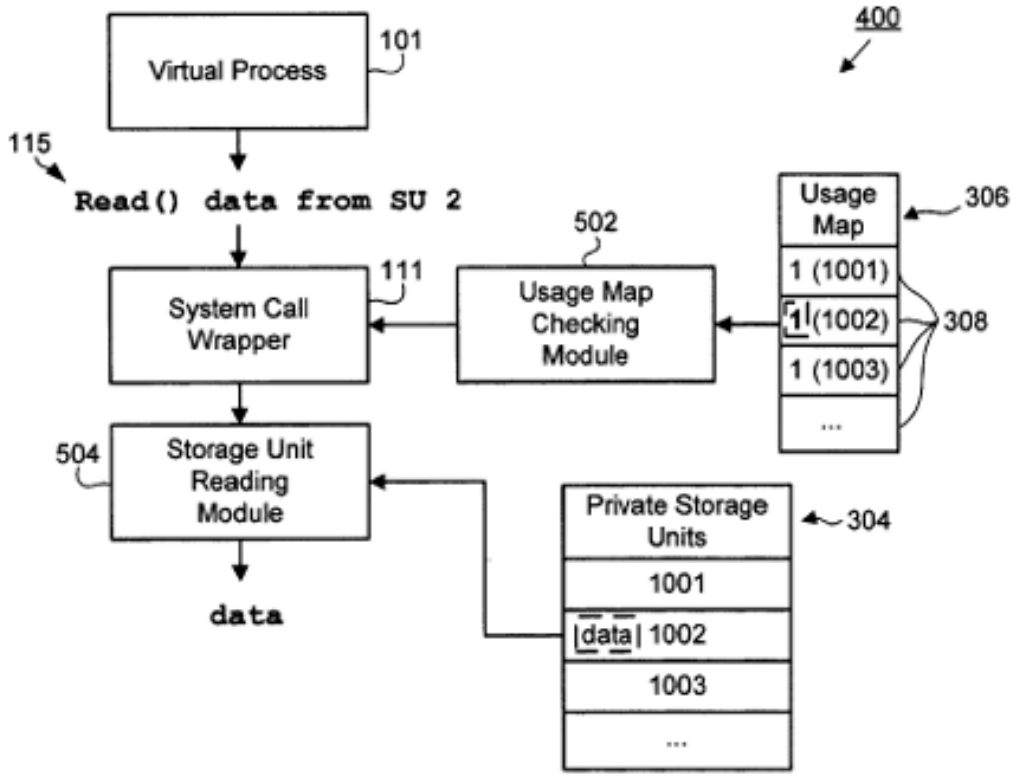


FIG. 5

Menage, Figure 5

240. As shown in Figure 5, "[a]fter the system call 115 for reading a data item is intercepted, a usage map checking module 502 checks for an indication 308 in the usage map 306 that the corresponding private storage unit 304 contains valid data," and "[i]f it does, a storage unit reading module 504 reads the data item from the private storage unit 304 rather than the shared storage unit 302." *Id.*, 9:35-41. However, as shown in Figure 6, "if the usage map checking module 502 determines that the usage map 306 does not include an indication 308 of valid data for the corresponding private storage unit 304, then the storage unit reading module 504 reads the data item from the shared storage unit 302 specified in the intercepted system call 115." *Id.*, 9:48-53, FIG. 6 (reproduced below).

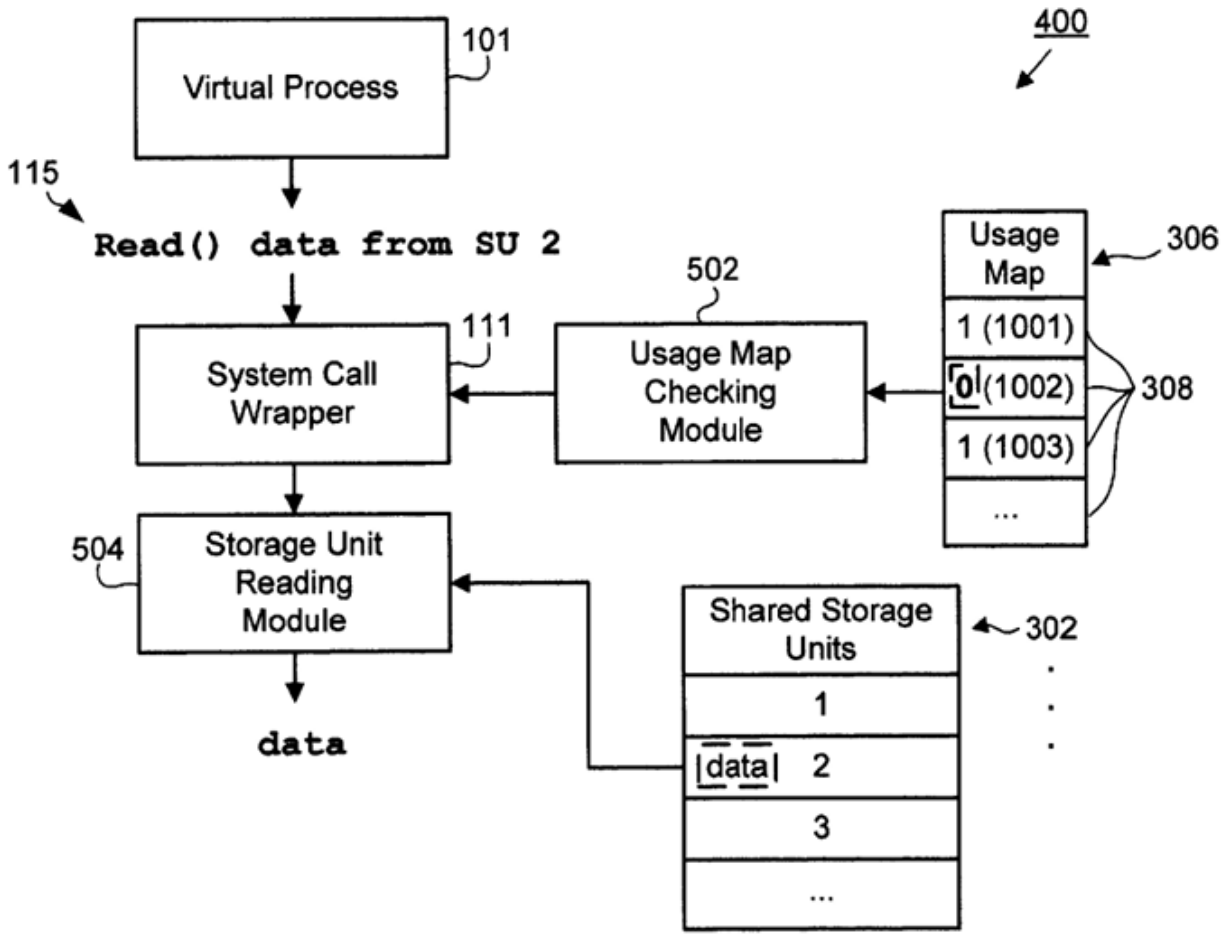


FIG. 6

Menage, Figure 6

241. As shown in Figure 7, "the file system 200 of each virtual process 101 may be conceptualized as a combination of the set of shared storage units 302 and the set of private storage units 304 from the corresponding template 310." *Id.*, 9:54-57, FIG. 7 (reproduced below).

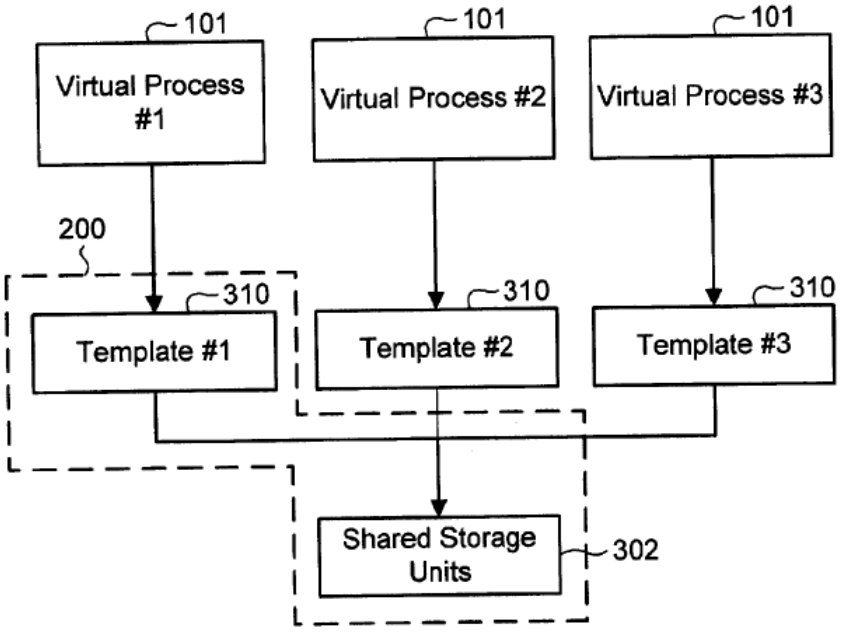


FIG. 7

Menage, Figure 7

242. In sum, "[e]ach attempt to write a data item to a shared storage unit 302 results in the data item being written to a private storage unit 304 in the template 310, with the usage map 306 being updated accordingly." *Id.*, 9:58-61. And, "[e]ach attempt to read a data item from a shared storage unit 302 results in the data item being read from a private storage unit 304 (if the private storage unit 304 contains valid data) or, instead, from the shared storage unit 302." *Id.*, 9:61-65.

243. Menage's "template-based approach to file system 200 creation and management is advantageous in a number of respects," for example, "there is no

need to duplicate an initial file system 200 for multiple virtual processes 101." *Id.*, 9:66-10:2. Menage also notes that "since all the virtual processes 101 use the same set of shared storage units 302, the shared storage units 302 may be easily cached for high speed access in the computer memory 102 or in a memory integrated with the storage device 204." *Id.*, 10:2-6.

B. Murphy

244. Murphy "relates generally to methods and apparatus maintaining computer systems," and more particularly, "the invention relates to systems for installing and maintaining files, file systems, BIOS data, operating system data and other data on computers in a networked environment." EX1005 (Murphy) at 1:10-14.

245. Murphy's Figure 1 depicts components typical in a personal computer system 100. *Id.*, 1:25-33, FIG. 1 (reproduced below).

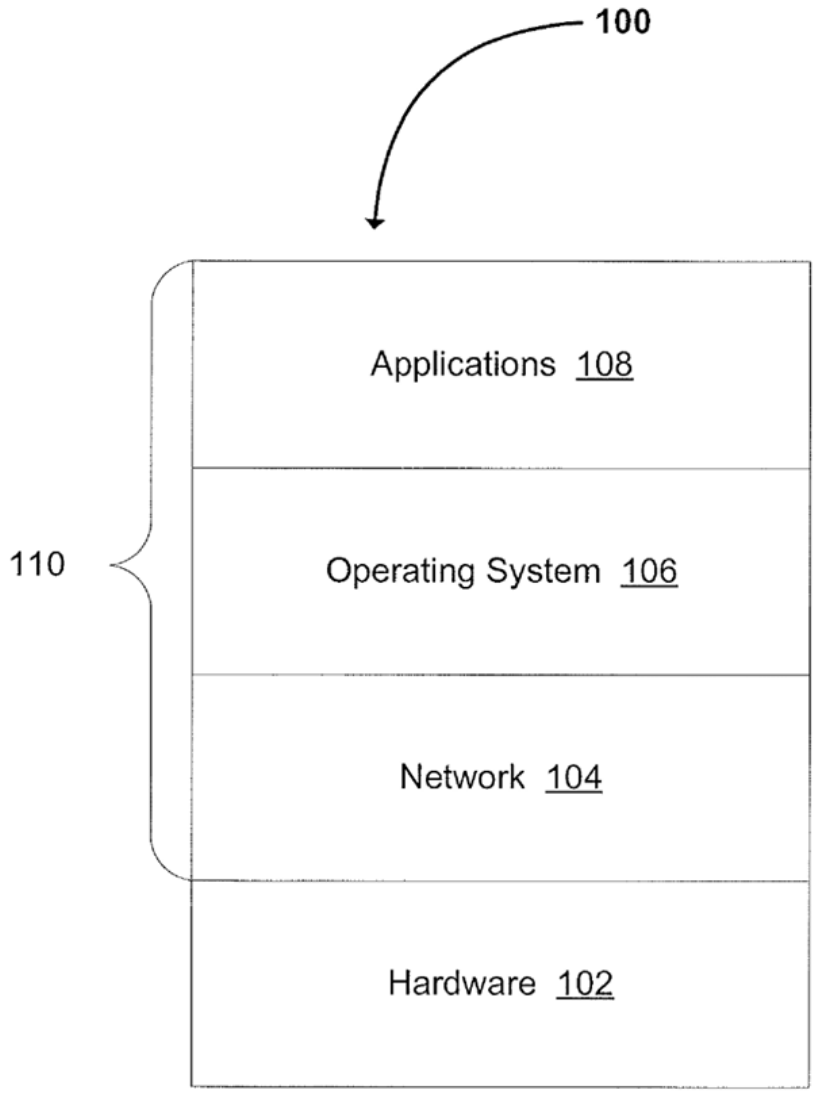


Figure 1
(prior art)

Murphy, Figure 1

246. Murphy explains that because "all aspects of computer system 100 are susceptible to failure, error, or the need for upgrade," that "an ideal administration program would be able to perform administrative tasks, repairs and upgrades on all

aspects of the computer system 100," operating "even if the network layer 104, operating system 106 or application layer 108 is not available." *Id.*, 1:39-49.

247. Murphy notes that although "various administration tools have existed in the past, these tools have exhibited one or more marked disadvantages," such as requiring "that hardware 102, network layer 104 and operating system 106 be available and functional." *Id.*, 1:50-60. Murphy also notes that "[c]onventional administrative programs frequently exhibit further disadvantages in that they typically function on a standalone PC, and as such are not suitable for use in a distributed, networked environment as required by most present day office environments," and that "[d]istributed administration of networked PCs is desirable because it is frequently cumbersome or inconvenient for support personnel to physically go to the computer to run administrative programs and to execute administrative tasks," which can require "a large amount of overhead in terms of time, effort and money." *Id.*, 1:63-2:9.

248. Murphy states that "[i]t is therefore desirable to create a method, system and apparatus to administer, maintain and upgrade computers from a central location," including where "problems exist with the computer's operating system or network interface." *Id.*, 2:10-14.

249. Murphy provides an overview of an exemplary remote "management and configuration system 200 [that] suitably includes one or more client computers

202 (also referred to herein as simply 'clients', examples of which are shown as 202A, 202B and 202C in FIG. 2) communicating via a network 210 with a server 206." *Id.*, 3:41-45, FIG. 2 (reproduced below).

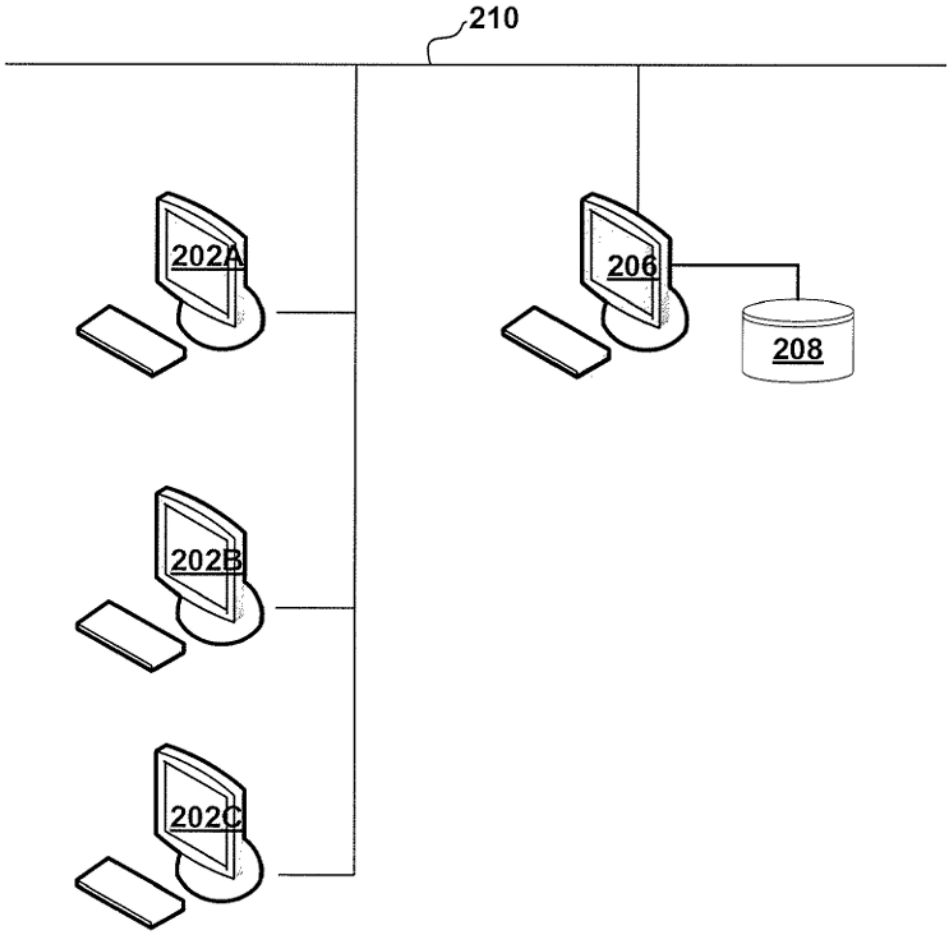


Figure 2

Murphy, Figure 2

250. In the Murphy system, "[a]s each client computer 202 is booted by a user or via the network, the client computer 202 may contact the server 206 to obtain configuration information." *Id.*, 3:46-48. Further, "[i]f server 206 recognizes client

computer 202 (e.g. if an entry exists in database 208), server 206 may send scripts or other configuration instructions to client computer 202 based upon events (such as calendar based or sequential events)." *Id.*, 3:54-58.

251. In Murphy, "[e]xemplary administrative tasks that may be carried out by scripts provided by server 206 to client 202 include, for example: booting the operating system directly from server 206, another server or a local drive; checking the integrity of the operating system, files, BIOS or other attributes of client 202; repairing or replacing the contents of the hard drive of client computer 202 with an image stored on server 206 or elsewhere; repairing or replacing files, directory entries, BIOS attributes and the like; and/or performing various other administrative or management tasks." *Id.*, 4:2-12.

252. Murphy's Figure 4A depicts "an exemplary pre-boot sequence" that "may allow one or more client computers 202 to obtain boot information from a server 206, and subsequently to obtain configuration information from server 206." *Id.*, 9:50-54, FIG. 4A (reproduced below).

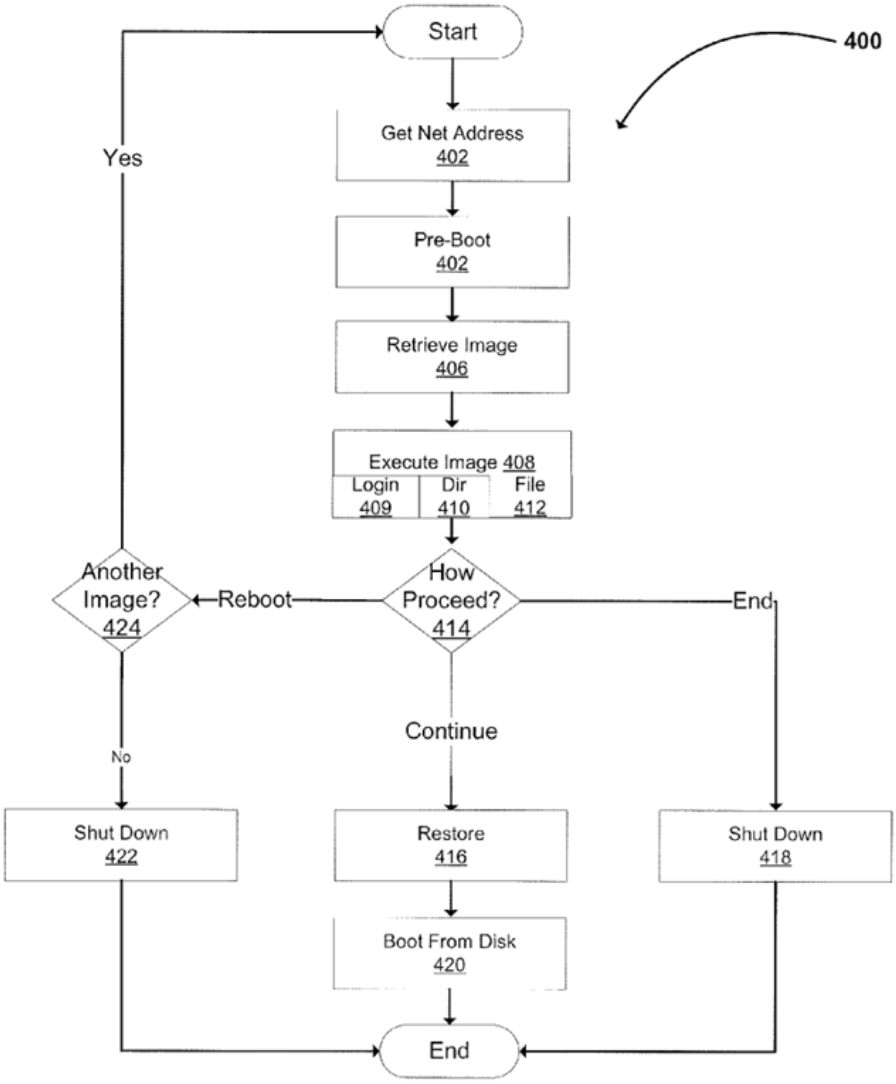


Figure 4A

Murphy, Figure 4A

253. At step 406 in Figure 4A, "[a] boot image may then be retrieved from server 206,"and the boot image "includes an operating system kern[e]l, a command interpreter and one or more configuration scripts (which may be executed by the operating system or the command interpreter)." *Id.*, 11:55-59. Murphy explains that

"other programs that may be included within the boot image or that may be obtained from server 206 include … a file management program 412." *Id.*, 11:37-59. Murphy's Figure 5 shows an exemplary "boot image 500," which includes a "bootable operating system (OS) 502" and a "file management program 412." *Id.*, 12:64-13:5, 14:16-19, FIG. 5 (reproduced below).

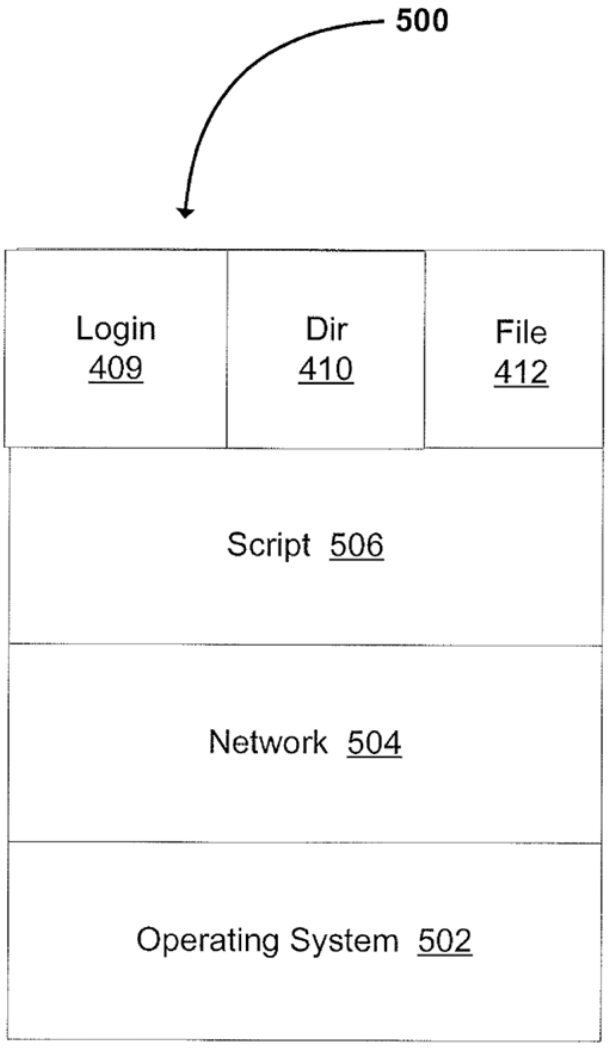


Figure 5

Murphy, Figure 5

254. In Murphy, the "[f]ile management program 412 may be used to manage file integrity on the local hard drive or another storage device affiliated with client 202." *Id.*, 14:50-52. For example, "a file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files." *Id.*, 14:52-56. This "index may … contain a listing of the relevant files, the size of the files, the file location and/or other information about the files." *Id.*, 14:56-58.

255. Murphy explains that "the index file may be created from the image file, which may be stored on server 206 or on client 202." *Id.*, 14:60-62. In one embodiment, "a single image file may be created for one or more client computers 202," which "may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance." *Id.*, 14:67-15:5.

256. I note that Murphy's "server 206" is erroneously referred to as "server 202" in the portion quoted in the previous paragraph. This is a typographical error, as the numeral "202" is used consistently throughout Murphy to refer to the "client computers 202," whereas the numeral "206" is used to refer to the "server 206." *See, e.g.*, *id.*, 3:41-45 ("With reference to FIG. 2, an exemplary management and configuration system 200 suitably includes one or more **client computers 202** … communicating via a network 210 with a **server 206.**"); *id.*, 14:60-62 ("[T]he index

file may be created from the image file, which may be stored on **server 206** or on **client 202**.").

C. Birse

257. Birse describes a system "for supplying a reliable and maintainable operating system in a net-booted environment. According to one embodiment, a network computer (NC) system including an NC server and multiple NC clients is managed by an NC client causing the remainder of the NC clients that are subsequently booted to receive operating system software that is configured differently than that currently in effect by replacing one or more system volumes on the NC server containing the operating system software with one or more different system volumes." EX1006 (Birse), Abstract.

258. In Birse, "[a]n 'image' refers to an electronic representation of an individual disk or a portion thereof." *Id*., 4:1-2.

259. In Birse, a "'thick client' generally refers to a computer system, such as a personal computer (PC), that includes a hard drive or other local storage medium. Traditionally, such computer systems boot and execute application programs from the local storage medium." *Id*., 4:22-26. Conversely, a "'thin client' generally refers to a disk-less computer system that relies upon external means, such as a server, to supply it with an operating system and application programs." *Id*., 4:27-30.

260. In Birse, "the terms 'network computer client' or 'NC client' generally refer to a client (thick or thin) that boots by accessing a copy of the operating system over a network. As such, an NC client is not required to be disk-less." *Id*., 4:31-35.

261. Moreover, a "'network computer server' or 'NC server' generally refer to a computer system that services requests of NC clients. For example, an NC server may provide access to a copy of the operating system to an NC client in response to a boot request." *Id*., 4:36-40.

262. Then, a "'network computer system' or 'NC system' refers to a network including one or more NC clients and one or more NC servers." *Id*., 4:41-43.

263. Birse's "'volume' generally refers to a fixed amount of storage on a storage medium, such as a disk or tape. In some instances, however, the term may be used as a synonym for the storage medium itself. However, it is possible for a single storage medium to contain more than one volume or for a volume to span more than one storage medium." *Id*., 4:44-50.

264. "According to one embodiment … a network computer ('NC') system maintains a copy of the OS that cannot be corrupted by ordinary users of the NC system." *Id*., 2:56-59.

265. "Additionally, the NC system may preserve user customizations (e.g., preferences) by maintaining individual, user, storage areas." *Id*., 2:59-61.

266. "When an NC client boots from the network and accesses a stored copy of the OS from an NC server, the user's preferences are dynamically merged with the system environment provided to the NC client." *Id*., 2:62-65.

267. Thus, because "the user's desktop preferences and other customized settings are preserved from session to session and supplied to the NC client as it boots from the network, the user may login to any NC client on the network and have the same user experience." *Id*., 2:65-3:3.

268. Birse's "FIG. 1 is a simplified block diagram that conceptually illustrates an NC system 100 according to one embodiment of the present invention" where" the NC system 100 includes an NC client 150 coupled to an NC server 170 via a data communication link 140, such as … Ethernet link." *Id*., 4:65-5:8, FIG 1 (reproduced below).

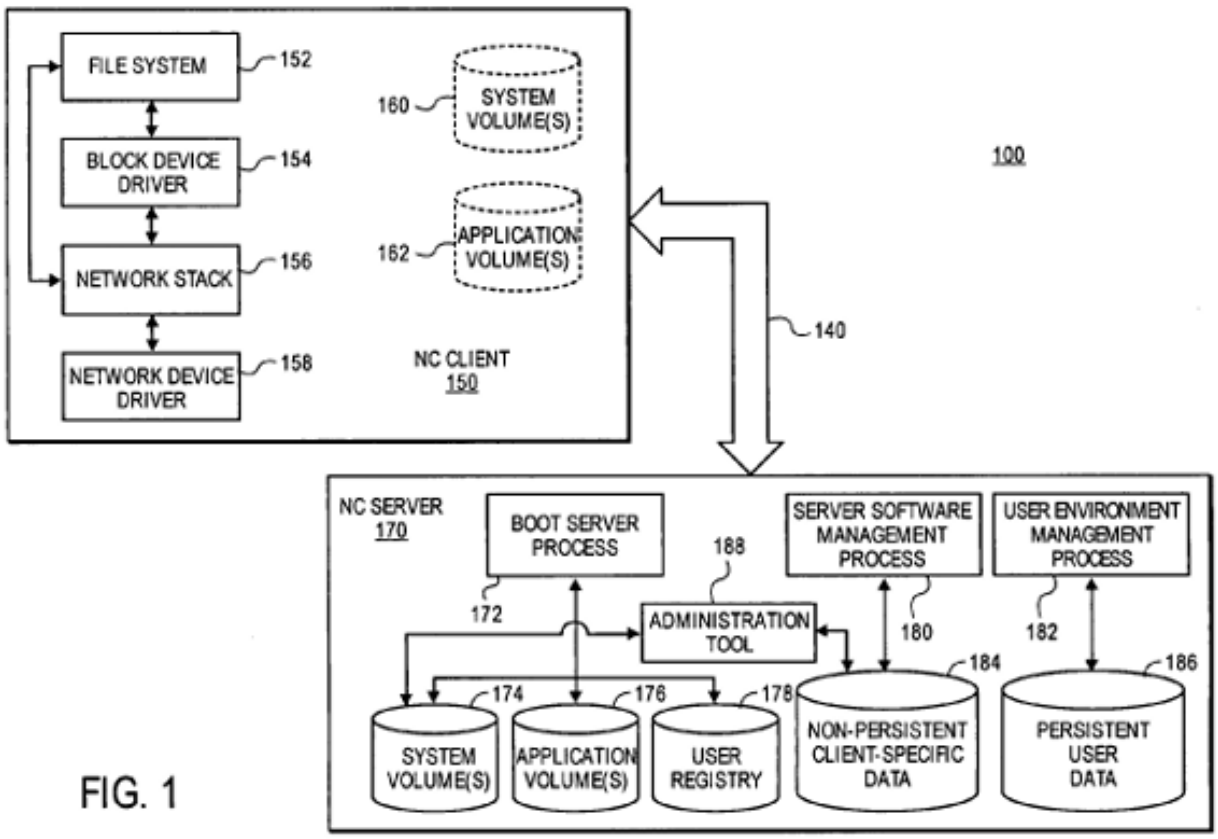


FIG. 1

Birse, Figure 1

269. "The NC server 170 includes one or more system volumes 174, one or more application volumes 176, a user registry 178, non-persistent client-specific data 184, and persistent user data 186." *Id*., 5:9-29.

270. Moreover, "[t]he system volumes 174 include a protected, read-only, master copy of the operating system software. The application volumes 176 include copies of various application programs for use by the NC client 150" and a "user registry 178 [that] is a database of authorized users, user passwords, NC client hardware addresses, such as Ethernet Media Access Control (MAC) addresses, and

NC client network addresses, such as Internet Protocol (IP) addresses." *Id*., 5:12-21.

271. Then, "[t]he non-persistent client-specific data 184 represents temporary data storage that is typically preserved only for the duration of a user session on a particular NC client. The persistent user data 186 represents long-term data storage for user information that is desirable to retain between user sessions, such as preferences, browser bookmarks, and other desktop environment customizations." *Id*., 5:9-29.

272. In Birse, "the boot server process 172 manages access to and from the system volumes 174 and the application volumes 176. Additionally, the boot server process 172 performs server-side bootstrapping processing" for example, using the "conventional Bootstrap Protocol (Bootp) server processing that waits for NC clients to appear on the network in accordance with Request For Comments (RFC) 951, … RFC 2132, and uses the standard extensions format." *Id*., 5:46-56.

273. "[T]o facilitate remote NC system administration, such as the installation of new system or application software, the boot server process 172 may serve application and system images that exist in the NC client's folder at the time of bootstrapping as a read-write file." *Id*., 6:4-8.

274. Birse's "server software management process 180 manages access to and from the non-persistent client-specific data 184. For example, at the conclusion

of a user session on an NC client, the server software management process 180 may provide data that is to be preserved between user sessions to the user environment management process 182 and reinitialize the client- specific storage area associated with the NC client for use by the next user." *Id*., 6:15-22.

275. Birse's "user environment management process 182 tracks and maintains the persistent user data 186 to insure that changes the user has made during the current session will be persistent at the next logic. For example, upon termination of a user session, preferences, desktop items, and various other information representing changes made by the user during the user session are copies to a user-specific storage location on the NC server 170." *Id*., 6:35-42.

276. Thus, "at the user's next login, from the same or different NC client, the user environment management process 182 will retrieve the data from the corresponding user-specific storage location and return it to the NC client thereby allowing the user to login to any NC client of the NC system 100 and have the same user experience." *Id*., 6:43-48.

277. Birse's "NC client also includes a file system 152, a block device driver 154, a network stack 156, and a network device driver 158, each of which may include hard-wired circuitry or machine-executable instructions or a combination thereof. Furthermore, at least a portion of such hard-wired circuitry and/or machine-executable instructions may be shared between a combination of the file system 152,

the block device driver 154, the network stack 156, and the network device driver 158." *Id*., 7:6-14.

278. The file system 152 represents logical structures and software routines that are used to control access to and from a local storage medium, such as a hard disk system." *Id*., 7:21- 23.

279. "The block device driver 154 services file system read and write requests to the system volumes 160." *Id*., 7:35-35.

280. The network stack 156 represents logical structures and software routines that are used to control access to and from the data communication link 140." *Id*., 7:55-57.

281. Birse discloses an NC client/server system that includes a base/root image (Core System Volume 922) and at least one leaf/incremental snapshot (shadow system volume 930. This is seen in Birse's Figure 9, reproduced below.

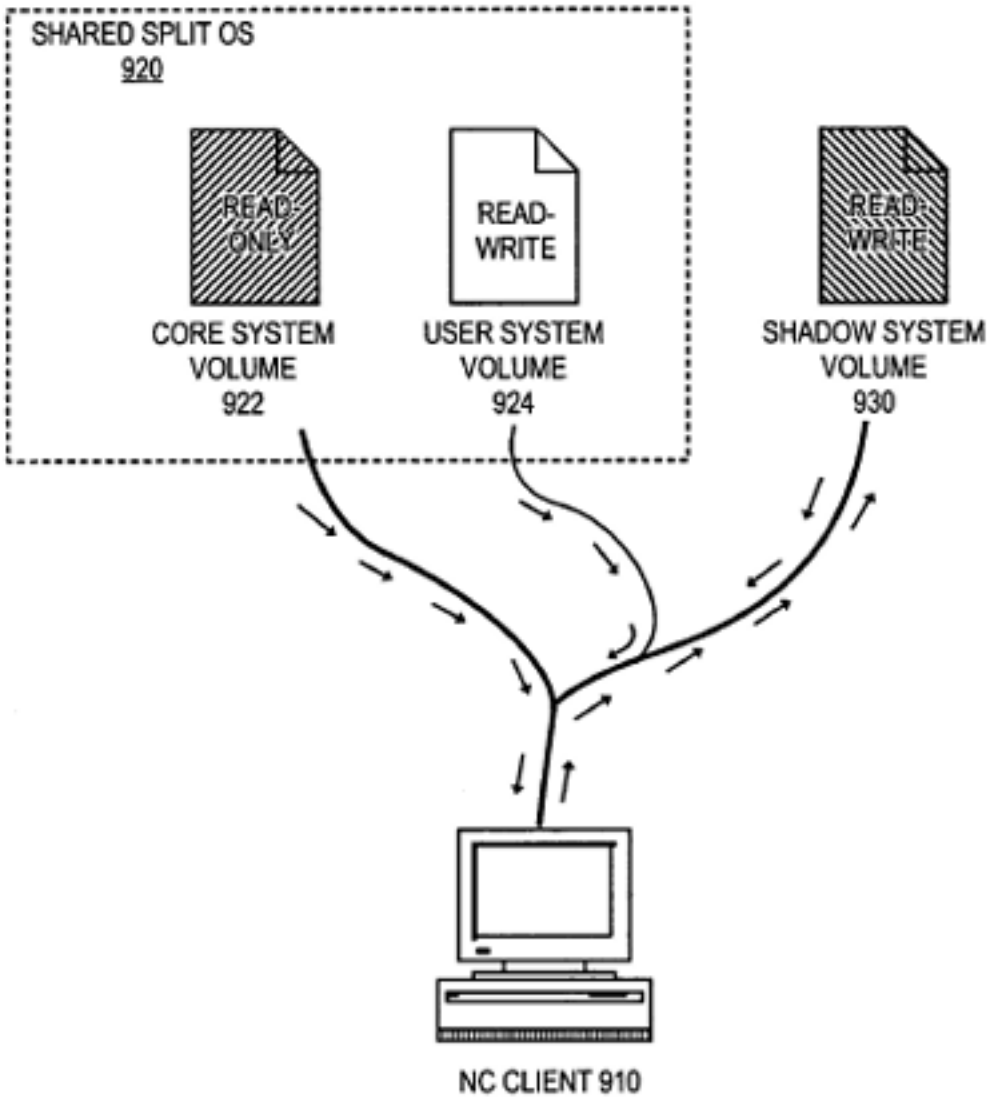


FIG. 9

Birse, Figure 9

282. "FIG. 9 is a block diagram that conceptually illustrates NC client interaction with a SplitOS 920 according to one embodiment of the present

invention. According to the example depicted, the SplitOS 920 of the NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924." *Id.*, 14:24-30.

283. Moreover, Birse states of Figure 9 that "[a]s indicated by the directional arrows along the connections between the core system volume 922, the user system volume 924, and the shadow system volume 930 and the NC client 910, data may be read from each of the core system volume 922, the user system volume 924, and the shadow system volume 930; however, data may only be written to the shadow system volume 930." *Id.*, 15:1-7. A POSITA would have understood Birse to manage snapshots as I describe in §VII.E above.

284. In Birse, "the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) 60 directed to the user system volume 930 are instead copied to the shadow system volume 930." *Id.*, 14:57-61. A POSITA would have understood Birse to manage snapshots using standard CoW techniques. *See also* §VII.E above.

285. The block device driver 154 services file system read and write requests to the system volumes 160. In one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver. However, in reality, the block device driver 154 may be configured to service the read and write requests

of the file system 152 by accessing the NC server 170." *Id.*, 7:35-41. A POSITA would have understood Birse to include a (logical/virtual) device driver that resides between file systems and storage devices, and merges/unifies I/O requests from applications. This functionality is an integral part of snapshot storage systems. *See* §VII.E above.

286. Birse discloses typical operating systems that use typical memory and caches. "The digital processing system 200 includes a processor 252, which may represent one or more processors and may include one or more conventional types of such processors, such as Motorola Power PC processor, an Intel Pentium (or x86) processor, etc. A memory 254 is coupled to the processor 252 by a bus 256. The memory 254 may be a dynamic random access memory (DRAM) and/or may include static RAM (SRAM). The processor may also be coupled to other types of storage areas/memories (e.g., cache, Flash memory, disk, etc.), which could be considered as part of the memory 254 or separate from the memory 254." *Id.*, 8:10-20.

287. Birse discloses that "the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) 60 directed to the user system volume 930 are instead copied to the shadow system volume 930." *Id.*, 14:57-61. A POSITA would have understood Birse's block driver to implement the functionality of

managing snapshots including merging their content, using standard CoW techniques. *See also* §VII.E above.

288. In sum, Birse discloses data structures and methods that a POSITA would have found functionally identical to the '844's UBD, especially in light of the '844 patent's description of what a UBD is and how it reportedly works (*see* §VIII.C above). Moreover, Birse operates and integrates with caches as discussed above. Lastly, a POSITA would have known that any computer systems, storage system, and more integrate with caches: that is, the software systems (*e.g.*, operating systems, applications) often consult the fast caches first, then the underlying (and typically slower layers), as I detail in §VII.D above.

D. Rothman

289. Rothman generally relates to "computer platforms and networks and, more specifically but not exclusively relates to techniques for performing network booting in an efficient manner." EX1007 (Rothman) at 1:6-9.

290. In Rothman, "boot images that are initially sent from a boot server to various clients are cached at network devices along communication paths between the boot server and the clients." *Id.*, Abstract. Subsequently, "[i]n response to subsequent boot image requests from the clients, boot images cached at the network devices are downloaded directly from the network devices rather than from the boot

server, reducing network traffic to and from the boot server and domains in between." *Id.*

291. Rothman's Figure 2 "shows a flowchart illustrating an overview of operations that are performed to support enhanced network boot efficiency, according to one embodiment." *Id.*, 3:63-65, FIG. 2 (reproduced below).

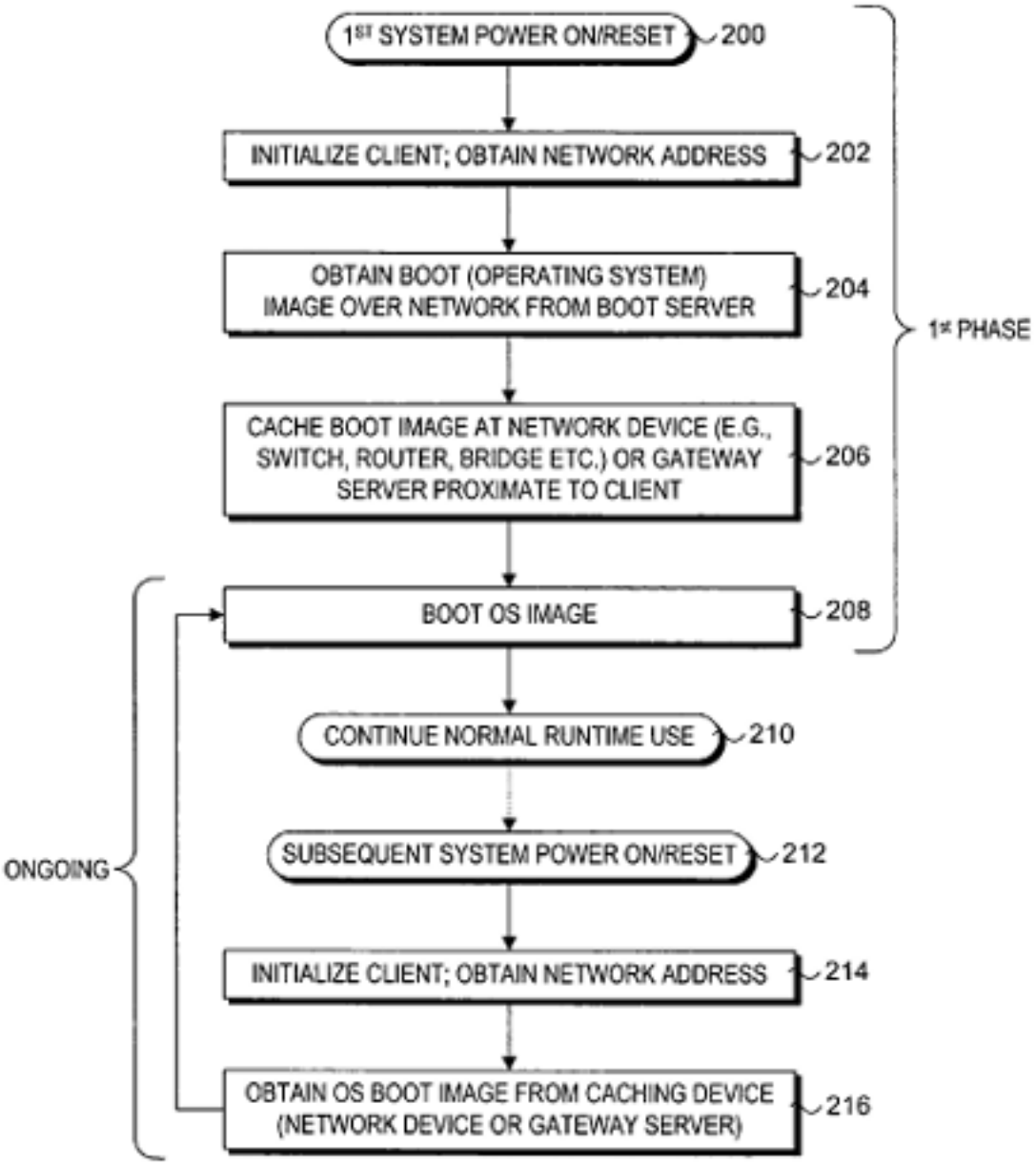


Fig. 2

Rothman, Figure 2

292. At block 200, "a first system power on event for a client" starts the "overall process," and at block 202, "the client obtains a network address." *Id.*, 3:65-

4:5. Then, at block 204, "the client obtains an operating system boot image over the network from a boot server." *Id.*, 4:6-7. At block 206, "[i]n conjunction with delivering the boot image to the client, the boot image is cached at a network device, such as a switch, router, bridge, etc., or a gateway server that is proximate to the client," and "[t]he image is then booted in a block 208 to enable run-time use, which is continued in a continuation block 210." *Id.*, 4:7-12. At block 212, "a subsequent system power on or reset event occurs," such as a reboot. *Id.*, 4:13-17. At block 214, a network address is obtained. *Id.*, 4:17-19. Then, at block 216 "the client obtains the OS boot image from the caching device (e.g., switch, router, bridge, gateway server, etc.) rather than the boot server." *Id.*, 4:19-22.

XI. CLAIM CONSTRUCTION

293. I understand that, for purposes of my analysis in this *inter partes* review proceeding, the terms appearing in the patent claims should be interpreted according to their "ordinary and customary meaning." In determining the ordinary and customary meaning, I understand that the words of a claim are first given their plain meaning that those words would have had to a person of ordinary skill in the art at the time of the alleged invention. I also understand that the structure of the claims, the specification, and file history may be used to better construe a claim. Moreover, I understand that even treatises and dictionaries may be used, albeit under limited

circumstances, to determine the meaning attributed by a person of ordinary skill in the art to a claim term.

294. As detailed in my analysis below, the challenged claims are obvious under any reasonable interpretation of the claim language.

295. For purposes of this declaration, I've been asked to assume that the time of the alleged invention is December 30, 2004 (the filing date of U.S. Patent Application 11/026,622, EX1012).

XII. DETAILED EXPLANATION OF GROUNDS

A. Ground 1: Menage Renders Claims 1-13 Obvious

1. Claim 1

a. [1.pre] A system for providing data to a plurality of compute nodes, the system comprising:

296. Menage teaches a system for providing data (combination of the set of shared and private storage units) to a plurality of compute nodes (virtual processes): "The present invention relates to file system creation and archival, particularly in the context of multiple virtual private servers running on a single physical host machine." EX1004 (Menage) at 3:56-58. As shown in Figure 7 (reproduced and annotated below), "the file system 200 of each virtual process 101 comprises a combination of the shared storage units 302 and the private storage units 304 from one or more templates 310." *Id.*, 8:63-67.

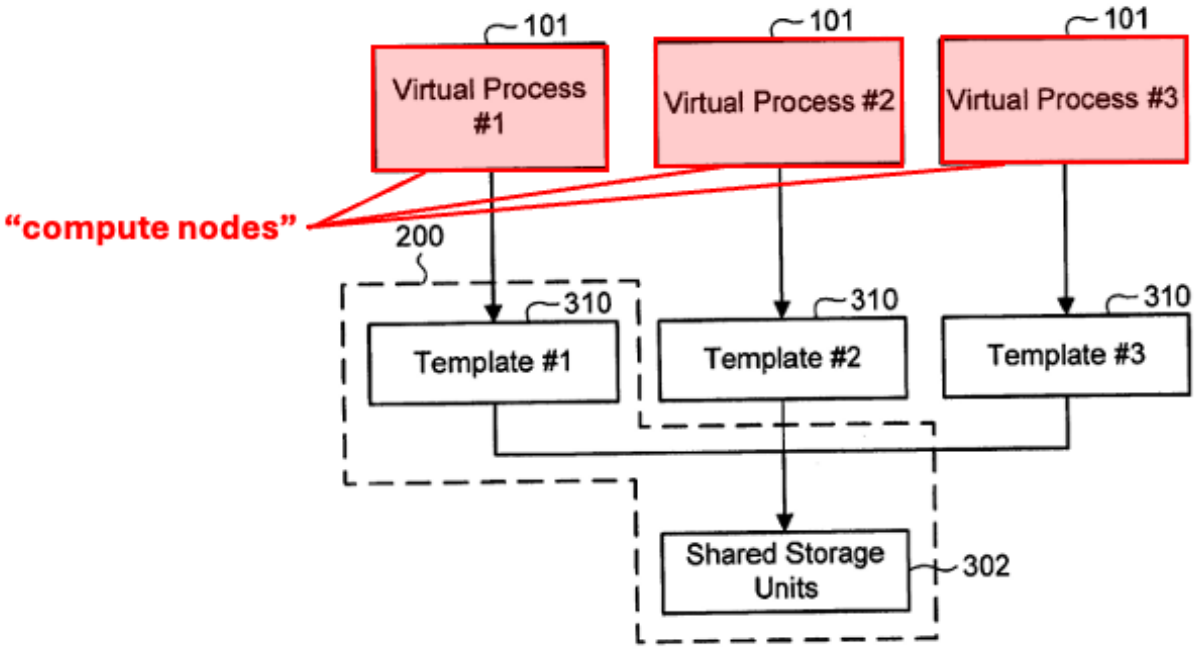


FIG. 7

Menage, Figure 7 (annotated)

297. Based on the disclosure of the '844 patent, a POSITA would have understood that "compute node" in this limitation refers to either a physical or virtual node that provides services, such as for an internet server. Notably, in the '844 patent, all embodiments are exemplary and there is nothing that expressly limits "compute node" to specific physical hardware. *See, e.g.*, *id.*, 1:35-37, 3:25-27, 4:26 ("Example Compute Node Operating Environment"). Without such a limitation in

the ’844 patent, and in view of the entirety of the disclosure, a POSITA would not understand that a “compute node” narrowly refers to only a separate physical node.

298. To the extent Patent Owner argues that “compute node” in this limitation and others in the ’844 patent refers to a separate, physical node, Menage still discloses and/or renders obvious [1.pre] under this narrow interpretation (which I disagree with). *See also* §§VII.A and VII.C above.

299. Menage explains in multiple places that it may be desirable to spread “virtual private servers” across a “comparatively small set of devices.” EX1004 (Menage), 2:3-6, 7:42-47. In those disclosed embodiments of Menage, each “device” running a virtual process would be a “compute node” under such an incorrectly narrow interpretation of the ’844 Patent.

300. It would also have been obvious to a POSITA that the “virtual processes” of Menage could have been distributed across multiple, physical nodes, such that the system would include multiple physical “compute nodes.” Distributing virtual processes across multiple physical devices would have achieved both scalability and fault tolerance (or high availability). For example, distributing virtual servers enables horizontal scaling, which allows the overall system to handle increased workloads through leveraging the combined resources of multiple physical machines. This approach also enhances system resilience because, if one physical node fails, virtual servers can be migrated or restarted on other available nodes,

minimizing downtime and ensuring continued availability of services. See §VII.C above.

301. In sum, Menage discloses "A system for providing data [the file system 200, which comprises the shared storage units 302 and the private storage units 304 from one or more templates] to a plurality of compute nodes [virtual processes 101]."

b. [1.a] a first storage unit configured to store blocks of a root image of said compute nodes;

302. Menage discloses a first storage unit (one or more units in the set of shared storage units 302) configured to store blocks of a root image (initial file system) of said compute nodes (virtual processes): "**[T]he set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101**, such as standard application programs, utilities, databases, etc. The set of shared storage units 302 forms **an initial file system 200** for each virtual process 101." EX1004 (Menage) at 8:10-15.

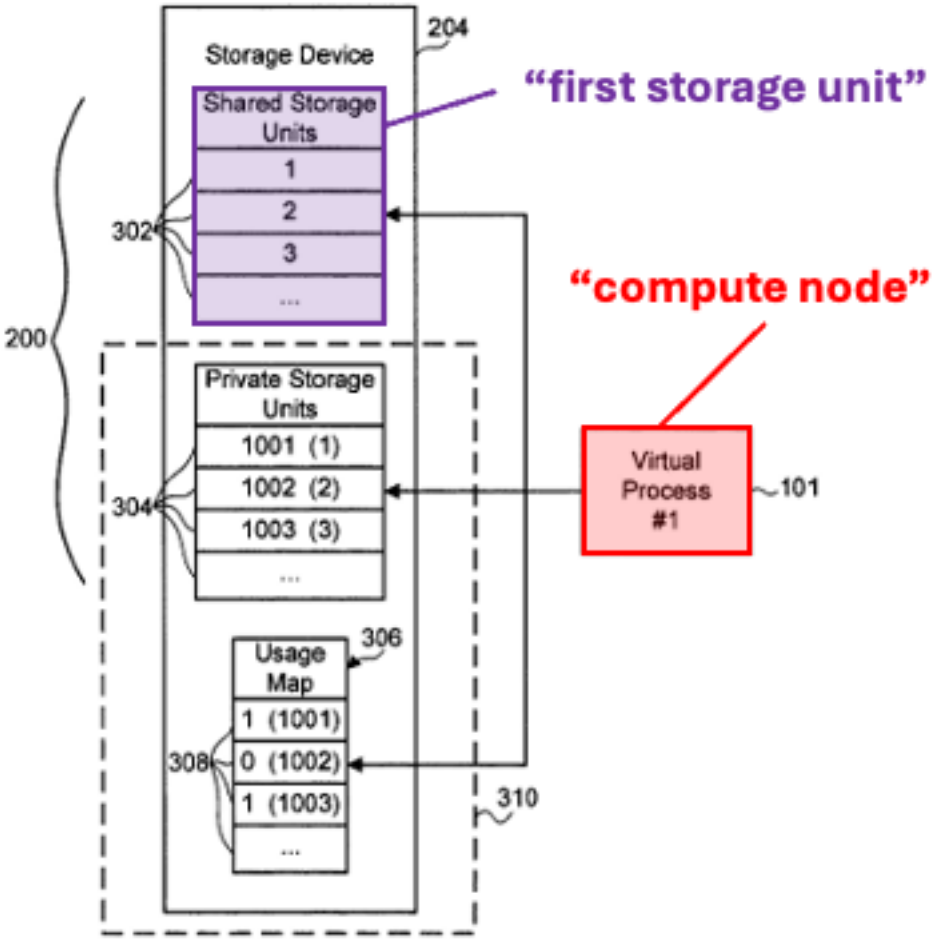


FIG. 3

Menage, Figure 3

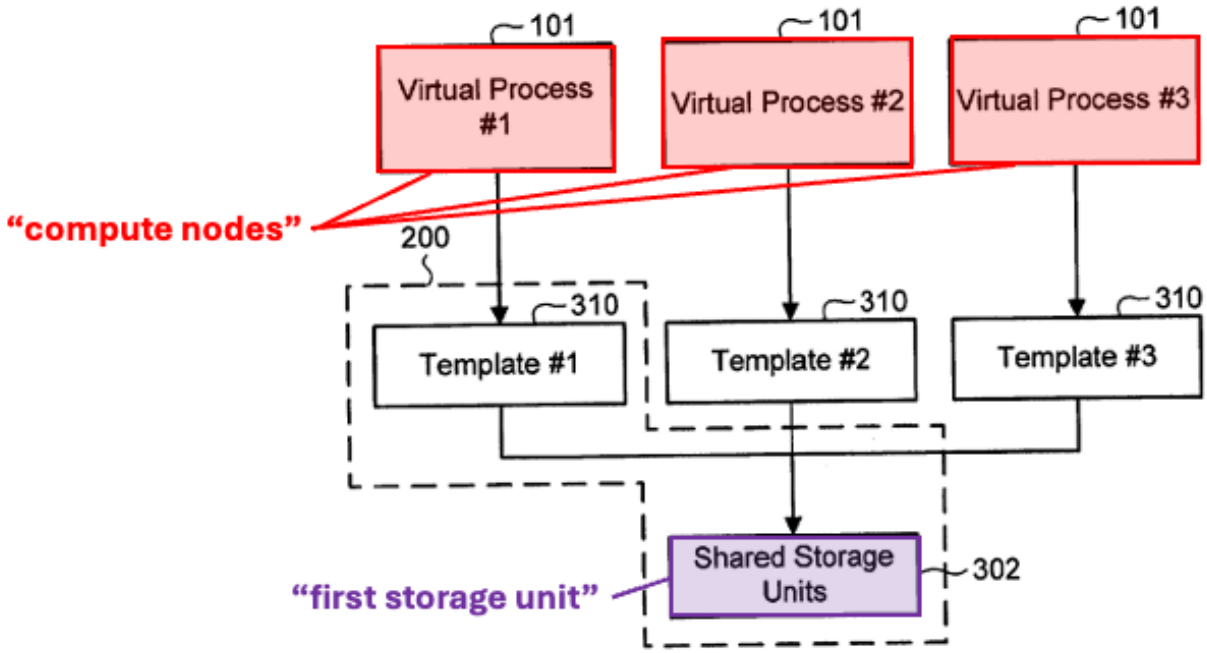


FIG. 7

Menage, Figure 7

303. I note that the claim recites a singular "storage unit." A POSITA would have understood that Menage's set represents a single unit/set, but also that every component or part of that set can itself be the claimed "storage unit."

304. According to Menage, "a file system is an organized accumulation of data within one or more physical storage devices, such as a hard disk drive or RAID (redundant array of inexpensive disks). The data is typically organized into 'files,' such as word processing documents, spreadsheets, executable programs, and the

like. The files are stored within a plurality of 'storage units' of the storage device, sometimes referred to as 'disk blocks' or 'allocation units.'" EX1004 (Menage) at 7:33-40.

305. A POSITA would have understood that, because Menage's "set of shared storage units 302 forms an initial file system 200 for each virtual process 101," and the files are stored within "disk blocks," Menage's set of shared storage units are configured to store blocks of the initial file system.

306. In sum, Menage discloses "a first storage unit [Menage's set of shared storage units 302, or parts thereof] configured to store blocks [Menage's disk blocks or allocation units] of a root image [Menage's initial file system 200] of said compute nodes [Menage's virtual processes 101]."

c. [1.b.i] a plurality of second storage units configured to store leaf images of respective compute nodes,

307. Menage discloses a plurality of second storage units (set of private storage units 304) configured to store leaf images of respective compute nodes (changes to the initial file system 200 that occur subsequent to the creation of the file system 200): "[A] set of private storage units 304 for a virtual process 101 and a corresponding usage map 306 are collectively referred to as a 'template' 310." EX1004 (Menage) at 8:58-61. "The set of private storage units 304, on the other hand, are used to store changes to the initial file system 200 that occur subsequent to the creation of the file system 200." *Id.*, 8:15-17.

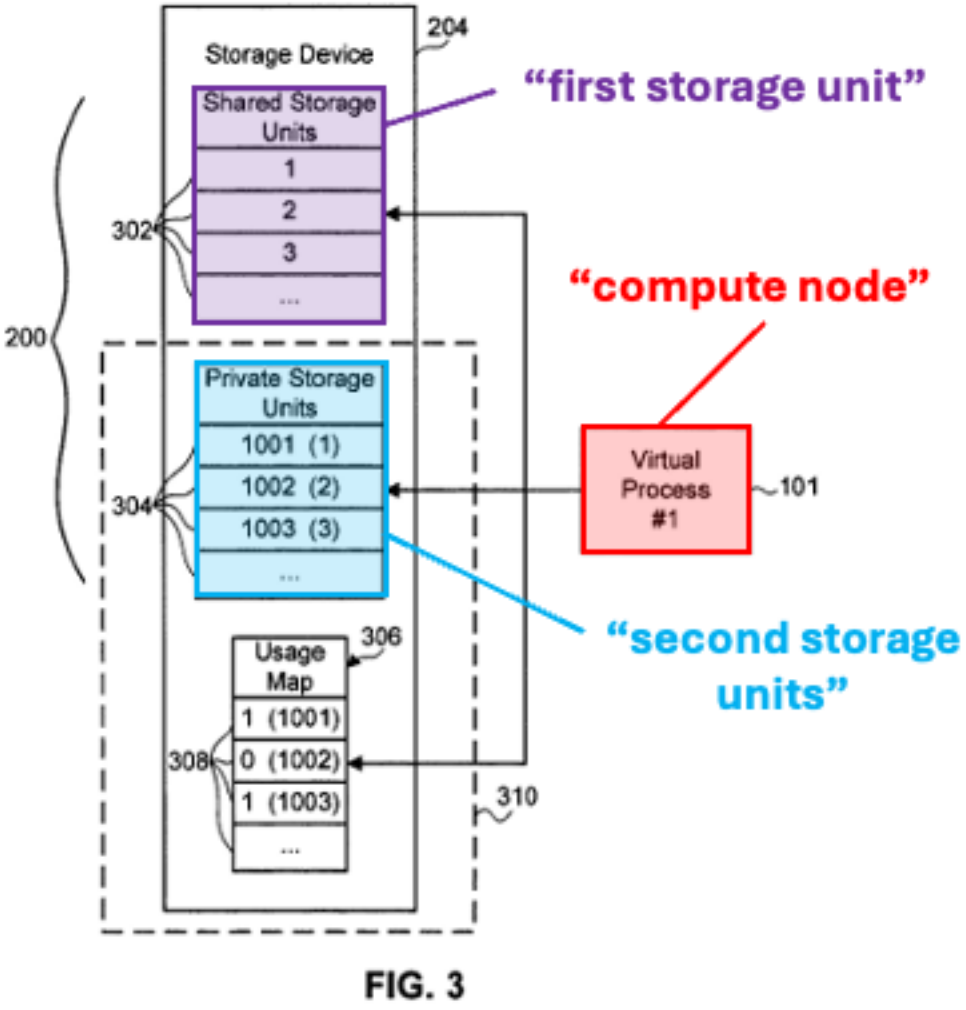


FIG. 3

Menage, Figure 3

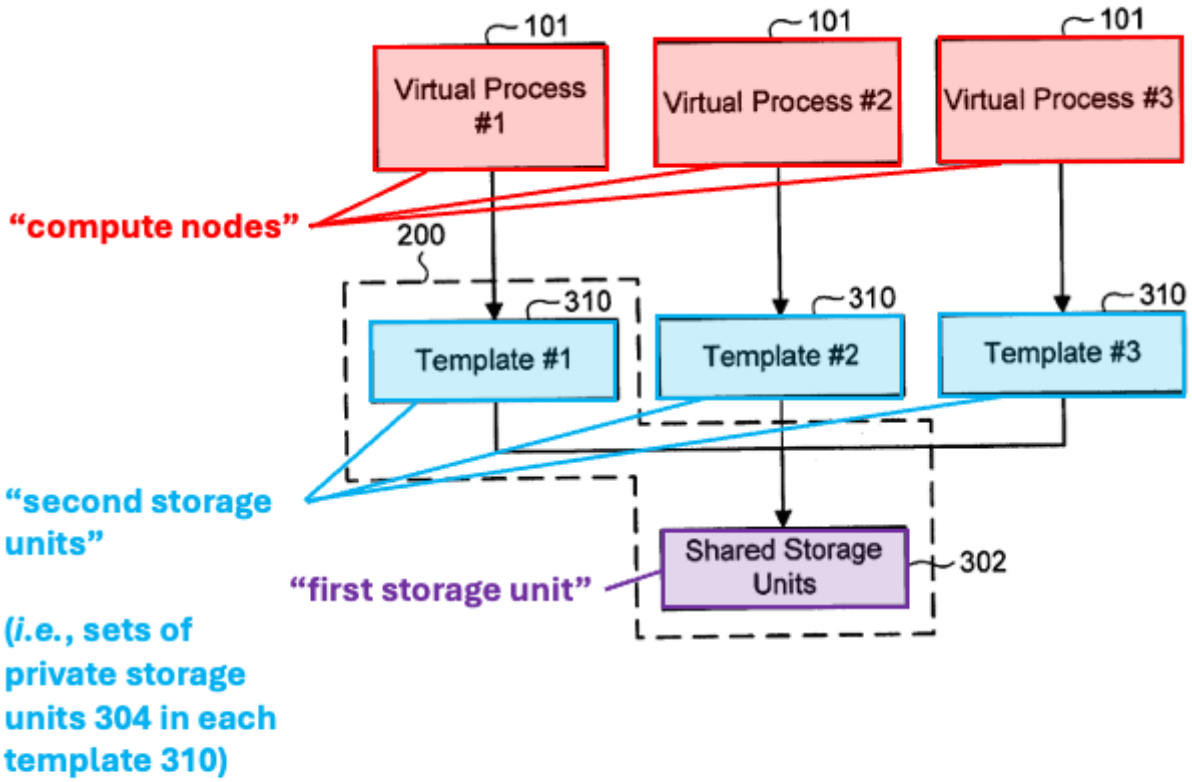


FIG. 7

Menage, Figure 7

308. According to Menage, "at least one template 310 is preferably provided for each virtual process 101," and "the file system 200 of each virtual process 101 comprises a combination of the shared storage units 302 and the private storage units 304 from one or more templates 310." *Id.*, 8:61-67. Menage explains that "[t]he set of private storage units 304 … **are used to store changes to the initial file system 200 that occur subsequent to the creation of the file system 200**." *Id.*, 8:16-18. And, "[e]ach attempt to write a data item to a shared storage unit 302 **results in the data item being written to a private storage unit 304 in the template 310**." *Id.*,

9:58-61. Thus, a POSITA would have understood that the private storage units 304 are configured to store leaf images (changes to the initial file system 200) of respective compute nodes (virtual processes 101).

309. In sum, Menage discloses "a plurality of second storage units [set of private storage units 304] configured to store leaf images of respective compute nodes [changes to the initial file system 200 to occur subsequent to the creation of the file system 200]."

d. [1.b.ii] said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

310. Menage discloses leaf images including only additional data blocks not previously contained in said root image (initial file system) and changes made by respective compute nodes to the blocks of said root image (changes to the initial file system 200 to occur subsequent to the creation of the file system 200), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes: "The set of private storage units 304, on the other hand, are used to store **changes to the initial file system 200 that occur subsequent to the creation of the file system 200**." EX1004 (Menage) at 8:16-18.

311. Menage teaches that "a usage map 306 is also provided for each set of private storage units 304. The usage map 306 includes a plurality of indicators 308 for indicating whether a corresponding private storage unit 304 contains valid data. As used herein, **the phrase 'contains valid data' means that the private storage unit 304 stores data representing a change to the initial file system 200**." *Id.*, 8:28-34.

312. As noted in §XII.A.1.c above regarding [1.b.i], "a set of private storage units 304 for a virtual process 101 and a corresponding usage map 306 are collectively referred to as a 'template' 310," and "at least one template 310 is preferably provided for each virtual process 101." *Id.*, 8:58-61. "[T]he usage map 306 is initially reset or 'initialized' to indicate that none of the private storage units 304 contain valid data. In the case of a bitmap embodiment, for example, all of the bits could be reset to 0." *Id.*, 8:46-49.

313. Regarding the process of writing data by one of the virtual processes 100 to the shared storage units 302, Menage teaches that "a system call wrapper 111, as described with reference to FIG. 1, intercepts a system call 115 for writing a data item to a shared storage unit 202." *Id.*, 9:4-7. "Rather than writing the data item to the shared storage unit 302, **however, a storage unit writing module 402 writes the data item to the corresponding private storage unit 304**." *Id.*, 9:10-12, Figure 4 (reproduced below).

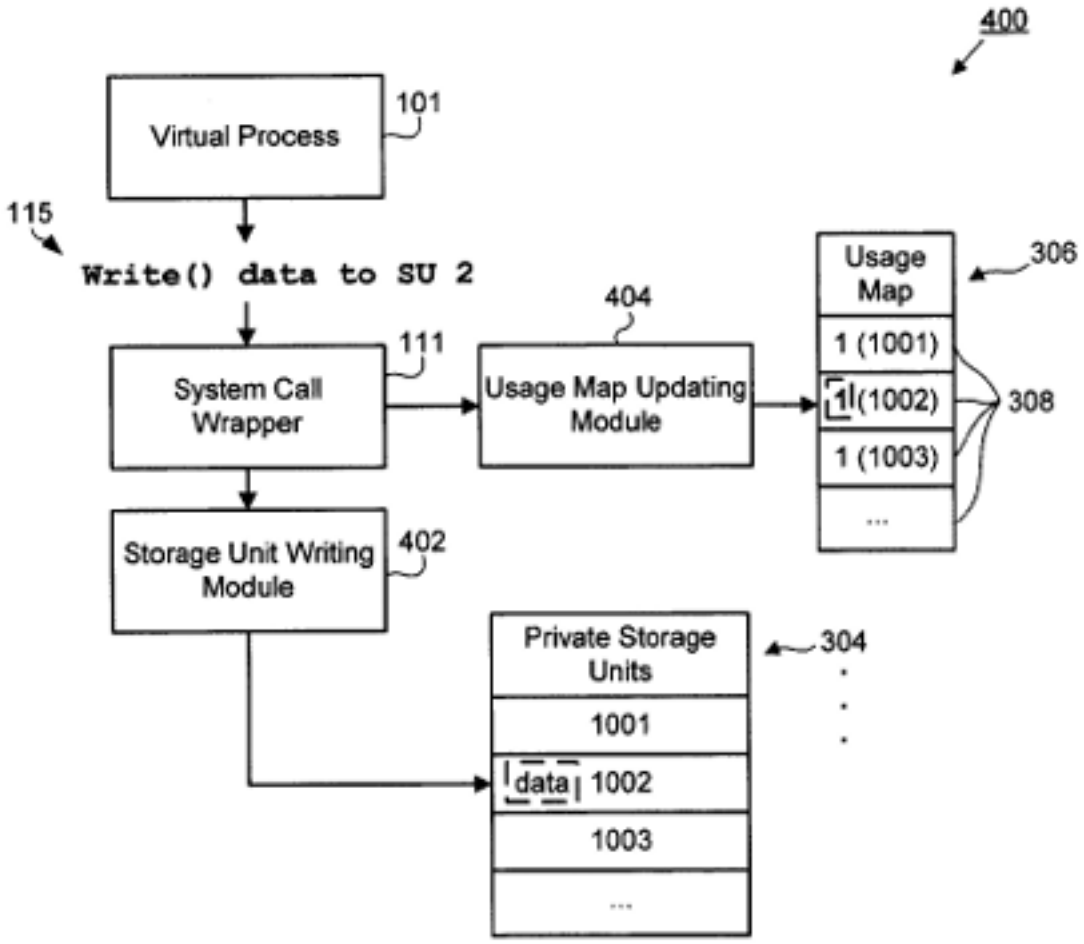


FIG. 4

Menage, Figure 4

314. Menage's "usage map 306" is then updated to indicate that the private storage unit 304 "**stores data representing a change to the initial file system 200**": "Additionally, a usage map updating module 404 stores an indication 308 in the usage map 306 that the private storage unit 304 (to which the data item is written) contains valid data. In the depicted embodiment, the usage map updating module

404 sets a bit in the usage map 306 that corresponds to the private storage unit 304." EX1004 (Menage) at 9:18-23.

315. Regarding the process of reading data by one of the virtual processes 100 from the shared storage units 302, Menage teaches that "the system call wrapper 111 is also configured, in one embodiment, to intercept a system call 115 for reading a data item from a shared storage unit 302." *Id.*, 9:30-33, Figure 5 (reproduced below).

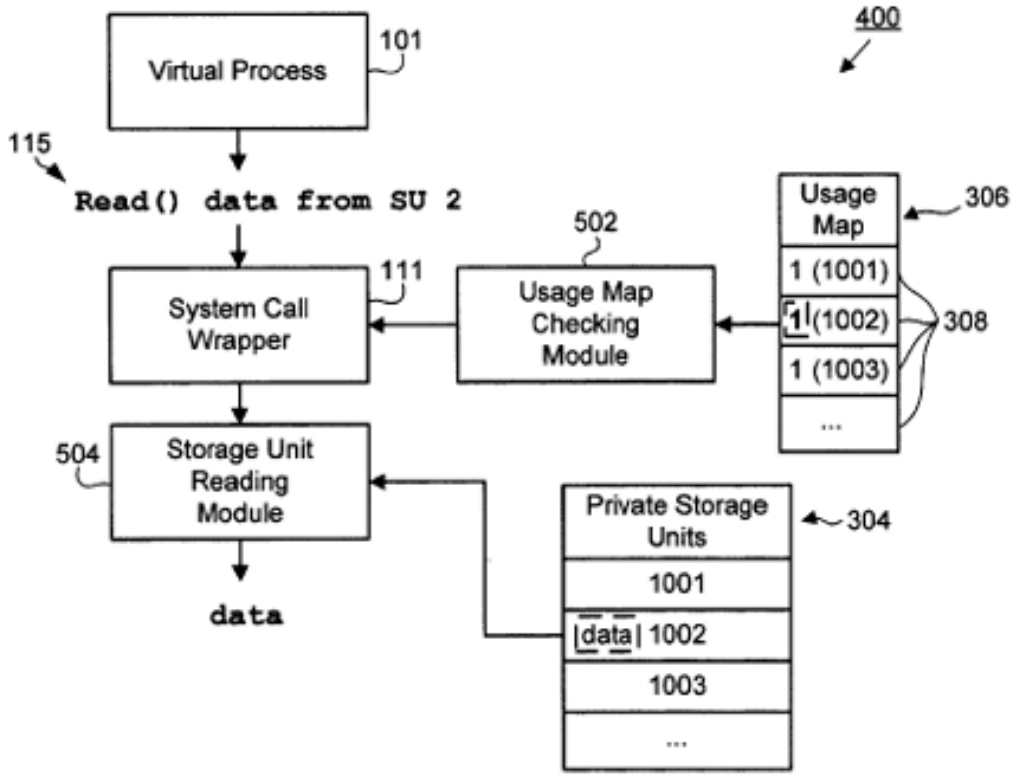


FIG. 5

Menage, Figure 5

316. If Menage's "usage map 306" indicates that the corresponding private storage unit 304 "contains valid data," *i.e.*, it "**stores data representing a change to the initial file system 200**," then the data is read from the private storage unit: "After the system call 115 for reading a data item is intercepted, a usage map checking module 502 checks for an indication 308 in the usage map 306 that the corresponding private storage unit 304 contains valid data. If it does, **a storage unit reading module 504 reads the data item from the private storage unit 304 rather than the shared storage unit 302**." *Id.*, 9:35-41.

317. In the event that the "usage map 306" *does not* indicate that the corresponding private storage unit 304 "contains valid data," *i.e.*, the data stored in the shared storage unit 302 has not been modified, the data is read from the shared storage unit: "As shown in FIG. 6, if the usage map checking module 502 determines that the usage map 306 **does not include an indication 308 of valid data** for the corresponding private storage unit 304, then **the storage unit reading module 504 reads the data item from the shared storage unit 302** specified in the intercepted system call 115." *Id.*, 9:48-53, Figure 6 (reproduced below).

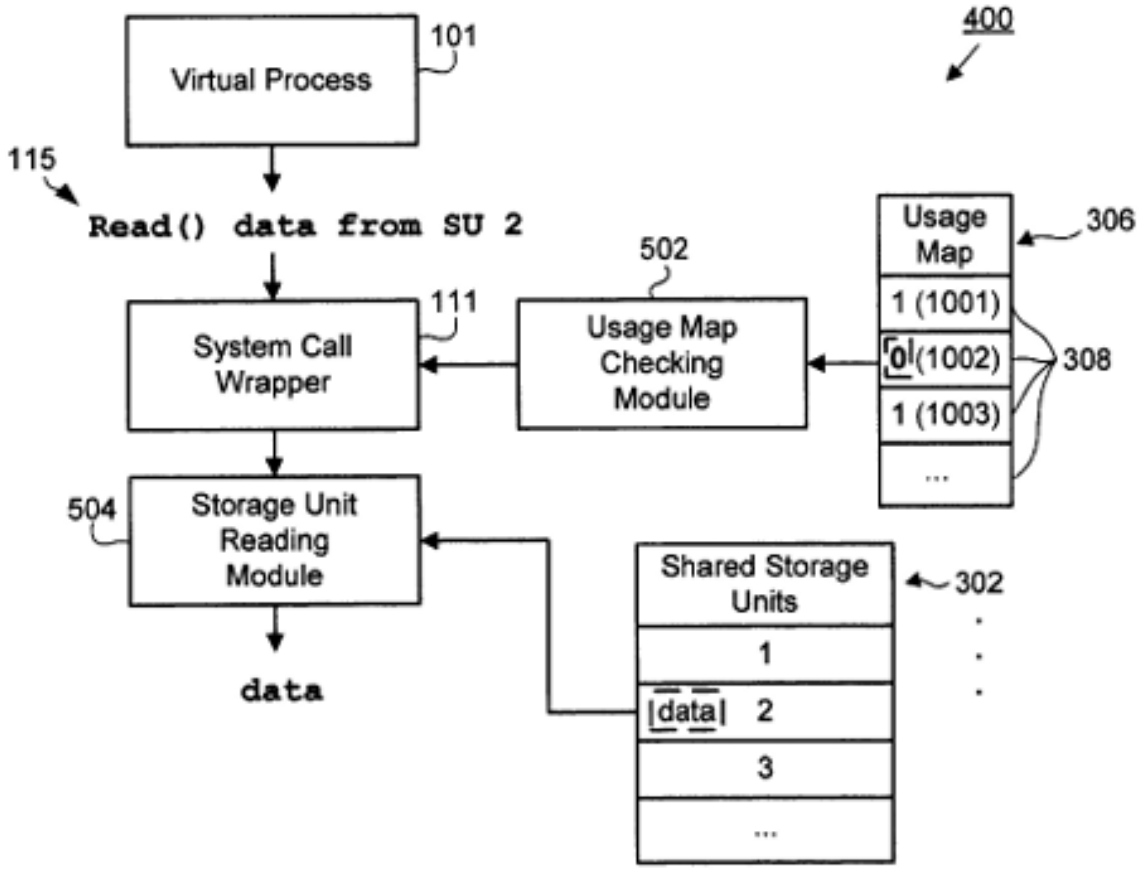


FIG. 6

Menage, Figure 6

318. A POSITA would have understood, therefore, that the private storage units include "only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image," as indicated by the usage map (*i.e.*, where "valid data" is indicated). Likewise, a POSITA would have understood that because data items are read from

the shared storage units when the usage map *does not* indicate that "valid data" is present in the private storage units, the "leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes." Finally, a POSITA would have recognized that Menage describes a process that is the same as reading data from one storage snapshot, to another snapshot, and so on, until reaching the base image/snapshot. *See also* §VII.E above.

319. In sum, Menage discloses "said leaf images including only additional data blocks not previously contained in said root image [initial file system 200] and changes made by respective compute nodes to the blocks of said root image [changes to the initial file system 200 to occur subsequent to the creation of the file system 200], wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes

e. [1.c] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

320. Menage discloses a cache (computer memory 102 or a memory integrated with the storage device 204) configured to cache blocks of said root image (initial file system) previously accessed by at least one of said compute nodes (virtual processes): "Moreover, since all the virtual processes 101 use the same set of shared storage units 302, **the shared storage units 302 may be easily cached for high**

speed access in the computer memory 102 or in a memory integrated with the storage device 204." EX1004 (Menage) at 10:2-6.

321. A POSITA would have recognized that blocks of the root image (the initial file system stored on the shared storage devices 204) would be cached upon access by at least one of the compute nodes (virtual processes), because this is in accordance with the basic operation of a cache memory, wherein data is admitted into the cache upon a read request for the data item. *See* §VII.D above.

322. In sum, Menage discloses "a cache [computer memory 102 or a memory integrated with the storage device 204] configured to cache blocks of said root image [initial file system 200] previously accessed by at least one of said compute nodes [virtual processes 101]."

2. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

323. *See* claim limitations [1.pre]-[1.c] in this Ground 1, in §§XII.A.1.a-XII.A.1.e above.

324. A POSITA would have understood that this limitation describes nothing more than common caching behavior (cache management policies, where caches naturally have limited room and must use thresholds to decide what to keep/evict).

325. Specifically, this limitation describes the "least-recently used" or "LRU" cache management policy. *See* §VII.D.1 above. As explained above, LRU is one of the oldest, most effective, and best-known algorithms. When the cache needs to make room for new items, LRU evicts enough items starting from the least-recently (*e.g.*, oldest) items first. *See* EX1012 (Silberschatz), 319-320 (describing the LRU algorithm). The '844 patent itself notes that removal based on "how recently the data was accessed" is but one "cache arbitration algorithms **known in the art**": "The data to be removed, deleted, overwritten, etc., may be selected the basis of, for example, how recently the data was accessed, how frequently the data has been accessed, or a combination of both. It should be appreciated that other cache arbitration algorithms known in the art may similarly be used." EX1001 ('844 patent) at 7:11-16.

326. It would have been obvious to a POSITA that the "cache threshold value" of "X" in this limitation is just the size of the cache (either the physical capacity of the cache memory or the size of a portion of memory allocated to be used as cache).

327. A simple example of the function of the LRU cache management policy follows. Assume an initially empty, fixed size cache of size "3," (depicted in the table below as 3 cache "slots") and a read sequence of arbitrary data items "A, B, C,

D, B, E." At each time step, the cache would contain the following data items (the last time the data was used is in parenthesis):

Time Step		**0**	**1**	**2**	**3**	**4**	**5**
Cache Slot	1	[**A(0)**]	[A(0)]	[A(0)]	[**D(3)**]	[D(3)]	[D(3)]
	2	[ ]	[**B(1)**]	[B(1)]	[B(1)]	[**B(4)**]	[B(4)]
	3	[ ]	[ ]	[**C(2)**]	[C(2)]	[C(2)]	[**E(5)**]

328. At time steps 0, 1, and 2, data items A, B, and C are added to the cache, respectively.

329. At time step 3, however, the LRU algorithm is applied to make room for the new data item D. Because A was last used at time step 0, it is the "least recently used" of the items in the cache, and is removed to make space to store D.

330. At time step 4, a request is made for a read of data item B. Because B is already in the cache, its last used time is updated to time step 4.

331. At time step 5, the LRU algorithm is again applied to make room for the new data item E. Because C was last used at time step 2 and B was re-used in time step 4, then item C is the "least recently used" of the items in the cache, and is removed to make space to store E.

332. Note that in each time step shown above, by removing the least recently used item from the cache, the cache necessarily stores the most recently accessed items. Thus, it would have been obvious to a POSITA, who would have had knowledge of the LRU cache management policy, "to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value."

333. In sum, Menage discloses that "said cache [computer memory 102 or a memory integrated with the storage device 204] is configured to store X most recently accessed blocks of said root image [initial file system 200], and wherein X represents a cache threshold value."

3. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

334. *See* claim limitations [1.pre]-[1.c] in this Ground 1, in §§XII.A.1.a-XII.A.1.e above.

335. Menage discloses that said first storage unit (shared storage units 302), said second storage units (private storage units 304), and said cache (computer memory 102 or a memory integrated with the storage device 204) are contained within a single storage appliance (storage device 204): "[T]he private storage units 304 are depicted as being located **within the same storage device 204** as the shared storage units 302." EX1004 (Menage) at 8:22-24. "[T]he shared storage units 302 may be easily **cached** for high speed access … **in a memory integrated with the storage device 204**." *Id.*, 10:2-6, Figure 3 (depicting shared storage units 302 and private storage units 304 as being located within the same storage device 204).

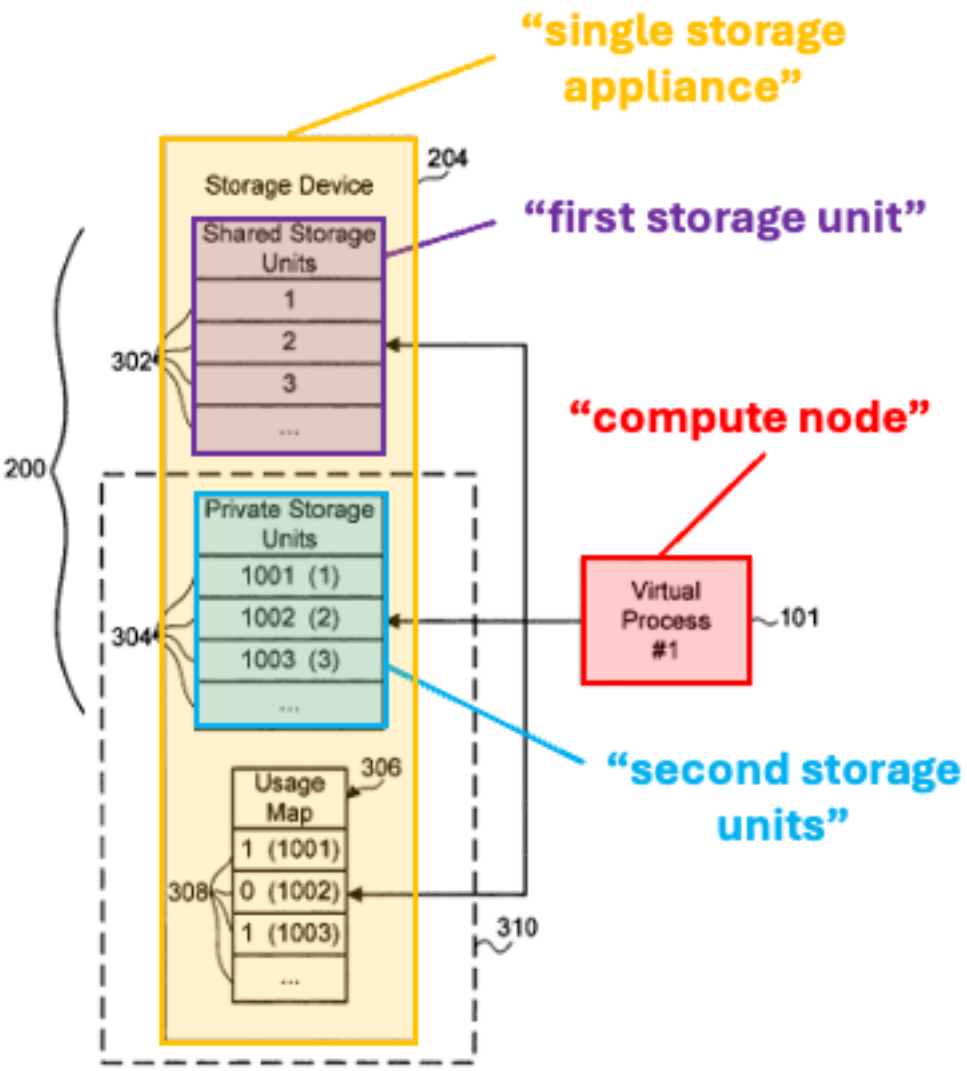


FIG. 3

Menage, Figure 3

336. In sum, Menage discloses that "said first storage unit [shared storage units 302], said second storage units [private storage units 304], and said cache [computer memory 102 or a memory integrated with the storage device 204] are contained within a single storage appliance [storage device 204]."

4. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

337. *See* claim limitations [1.pre]-[1.c] in this Ground 1, in §§XII.A.1.a-XII.A.1.e above.

338. The term "union block device" is informally defined in the '844 patent by way of the functions that it performs. *See* §VIII.C above. The '844 patent explains that "[i]n one embodiment, UBDs 230*a-n* are effectively low-level drivers that operate as an interface between the first and second storage devices and the file system of each compute node 220*a-n*." EX1001 ('844 patent) at 5:60-64. However, I note that there is no requirement in claim [4] that the "union block devices" be low-level drivers (which is required by claim [5], discussed in §XII.A.5 below).

339. Menage discloses a plurality of union block devices (system call wrapper(s), together with a storage unit writing module, a storage unit reading module, a usage map updating module, and a usage map checking module) configured to interface between respective compute nodes (virtual processes) and said first storage unit (shared storage units), respective second storage units (private storage units), and said cache to distribute application environments by merging the blocks of said root image (initial file system 200) with the blocks of respective leaf

images (changes to the initial file system 200 to occur subsequent to the creation of the file system 200): "the set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101, such as standard application programs, utilities, databases, etc.," and "forms an initial file system 200 for each virtual process 101." EX1004 (Menage) at 8:10-15. Conversely, the "set of private storage units 304 … are used to store changes to the initial file system 200 that occur subsequent to the creation of the file system 200." *Id.*, 8:16-18.

340. As explained in §XII.A.1.d above with respect to claim limitation [1.b.ii], the system call wrapper, together with a storage unit writing module, a storage unit reading module, and a usage map checking module, merge the changes recorded on the private storage units with the "initial file system" that is stored in the set of shared storage units by directing requests to the shared storage units and private storage units based on whether the usage map contains "valid data." The resultant, complete file system that is created for each virtual process, by the merging the changes to the "initial file system" stored in the private storage units with the data in the shared stored units, is an "application environment" as required by claim [4].

341. In sum, Menage discloses "a plurality of union block devices [system call wrappers 111, together with a storage unit writing module 402, a storage unit reading module 504, a usage map updating module 404, and a usage map checking

module 502] configured to interface between respective compute nodes [virtual processes 101] and said first storage unit [shared storage units 302], respective second storage units [private storage units 304], and said cache [computer memory 102 or a memory integrated with the storage device 204] to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image [initial file system 200] with the blocks of respective leaf images [changes to the initial file system 200 to occur subsequent to the creation of the file system 200]."

5. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

342. *See* claim [4] in this Ground 1, in §XII.A.4 above.

343. As explained with respect to claim limitation [1.b.ii] (§XII.A.1.d above) and claim [4] (§XII.A.4 above), Menage teaches a plurality of union block devices (system call wrapper, together with a storage unit writing module, a storage unit reading module, and a usage map checking module) that intercept read and write requests to the shared storage units and redirect the operations to or from the private storage units as necessary.

344. Menage discloses that the union block devices (system call wrapper, together with a storage unit writing module, a storage unit reading module, and a

usage map checking module) comprise low-level drivers (virtual disk device) for interfacing between the file systems of respective compute nodes (virtual processes) and said first storage unit (shared storage units), respective second storage units (private storage units), and said cache: "[I]nterception of system calls is illustrated as a primary vehicle for file system virtualization. However, virtualization at other levels is also contemplated. For example, **virtualization may take place at a lower level, such as within a virtual disk device**. In such an embodiment, the virtualization may occur once the file system code has turned a system call request into **requests for particular blocks on the virtual block device**." EX1004 (Menage) at 12:15-22.

345. It would have been obvious to a POSITA that a "virtual disk device" that performs the same functions as the system call wrapper and associated modules is a "low-level driver" as required by claim [5].

346. In sum, Menage discloses that "said union block devices [system call wrappers 111, together with a storage unit writing module 402, a storage unit reading module 504, a usage map updating module 404, and a usage map checking module 502] comprise low-level drivers [virtual disk devices] for interfacing between the file systems of respective compute nodes [virtual processes 101] and said first storage unit [shared storage units 302], respective second storage units [private

storage units 304], and said cache [computer memory 102 or a memory integrated with the storage device 204]."

6. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only.

347. *See* claim limitations [1.pre]-[1.c] in this Ground 1, in §§XII.A.1.a-XII.A.1.e above.

348. As explained in §XII.A.1.d above with respect to claim limitation [1.b.ii], Menage teaches that "a system call wrapper 111, as described with reference to FIG. 1, intercepts a system call 115 for writing a data item to a shared storage unit 202." EX1004 (Menage) at 9:4-7. "**Rather than writing the data item to the shared storage unit 302**, however, a storage unit writing module 402 **writes the data item to the corresponding private storage unit 304**." *Id.*, 9:10–12.

349. It would have been obvious to a POSITA that the shared storage unit 302 is read-only, given that all write requests that are directed to the shared storage unit 302 are instead redirected to the private storage unit 304.

350. In sum, Menage discloses that "said first storage unit [shared storage units 302 does not get written to] is read-only."

7. Claim 7

a. [7.pre] A method for providing data to a plurality of compute nodes, comprising:

351. *See* claim limitation [1.pre] in this Ground 1, in §XII.A.1.a above.

b. [7.a] storing blocks of a root image of said compute nodes on a first storage unit;

352. *See* claim limitation [1.a] in this Ground 1, in §XII.A.1.b above.

c. [7.b.i] storing leaf images for respective compute nodes on respective second storage units,

353. *See* claim limitation [1.b.i] in this Ground 1, in §XII.A.1.c above.

d. [7.b.ii] said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

354. *See* claim limitation [1.b.ii] in this Ground 1, in §XII.A.1.d above.

e. [7.c] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

355. *See* claim limitation [1.c] in this Ground 1, in §XII.A.1.e above.

8. Claim 8: The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and providing said first portion of said data to said at least one of said compute nodes from said cache memory.

356. *See* claim limitations [7.a]-[7.c] in this Ground 1, in §§XII.A.7.a-e above.

357. As explained with respect to claim limitation [1.c] in §XII.A.1.e above, Menage discloses a cache: "Moreover, since all the virtual processes 101 use the

same set of shared storage units 302, the shared storage units 302 may be easily cached for high speed access in the computer memory 102 or in a memory integrated with the storage device 204." EX1004 (Menage) at 10:2-6.

358. It would have been obvious to a POSITA that the additional limitations of claim [8] describe nothing more than the standard operation of a cache memory. The "high speed access" mentioned by Menage is achieved because read requests for the data items that are in the cache memory are provided from the cache, which is faster than the underlying data store. *See* §VII.D above.

359. In sum, Menage discloses receiving a read request from at least one of said compute nodes [virtual processes 101], wherein a first portion of the data requested is currently stored in said cache memory [computer memory 102 or a memory integrated with the storage device 204]; and providing said first portion of said data to said at least one of said compute nodes from said cache memory."

9. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

360. *See* claim [8] in this Ground 1, in §XII.A.8 above.

361. As explained with respect to claim limitation [1.c] in §XII.A.1.e above, Menage discloses a cache: "Moreover, since all the virtual processes 101 use the same set of shared storage units 302, the shared storage units 302 may be easily

cached for high speed access in the computer memory 102 or in a memory integrated with the storage device 204." EX1004 (Menage) at 10:2-6.

362. It would have been obvious to a POSITA that the additional limitations of claim [9] describe nothing more than the standard operation of a cache memory. In the event of a cache miss, the data requested is retrieved from the backing store and stored in the cache for subsequent faster reading from the cache. Upon a subsequent request for the same data (and provided that the data has not been evicted from the cache), the data is retrieved from the faster cache rather than the backing store. *See* §VII.D above.

363. In sum, Menage discloses "updating said cache memory [computer memory 102 or a memory integrated with the storage device 204] based on said read request."

10. Claim 10: The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

364. *See* claims [2] (§XII.A.2 above) and [9] (§XII.A.9 above) in this Ground 1.

11. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

365. *See* claim [4] (§XII.A.4 above) and claim limitations [7.a]-[7.c] (§§XII.A.7.b-e above) in this Ground 1.

12. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

366. *See* claims [5] (§XII.A.5 above) and [11] (§XII.A.11 above) in this Ground 1.

13. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only.

367. *See* claim [6] (§XII.A.6 above) and claim limitations [7.a]-[7.c] (§§XII.A.7.b-e above) in this Ground 1.

B. Ground 2: Menage in View of Murphy Renders Claims 14-27 Obvious

1. Motivation to Combine Menage with Murphy

368. A POSITA would have been motivated to combine Menage and Murphy for at least the following five reasons.

369. **First**, the combination of Menage and Murphy represents the use of known techniques according to known methods in the same way to yield predictable results. Menage and Murphy both describe systems wherein multiple systems share a base file system. For example, in Menage, the "virtual processes 101" share a "set

of shared storage units 302." Likewise, in Murphy, the "client computers 202" may share a single "boot image 500."

370. The Menage and Murphy combination would merely involve using Murphy's indexing in Menage's system, both of which were known, according to Murphy's methods for creation of an index for a boot image. For example, Menage teaches the use of a "set of shared storage units 302" to store data that is common to multiple virtual processes, and forms "an initial file system." *See* EX1004 (Menage) at 8:10-15 ("[T]he set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101, such as standard application programs, utilities, databases, etc. The set of shared storage units 302 forms an initial file system 200 for each virtual process 101.").

371. Murphy describes the creation of an index file from a boot image that may be shared by multiple client computers. *See* EX1005 (Murphy) at 14:60-62 ("[T]he index file may be created from the image file, which may be stored on server 206 or on client 202."); *id.*, 14:67-15:5 ("[A] single image file may be created for one or more client computers 202 [that] may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance.").

372. A POSITA would have recognized the similarity between Menage's "shared storage units" and Murphy's "single image" file for use with "multiple client

computers," and therefore would have found it obvious to create an index (as taught by Murphy) for the "shared storage units" in Menage's system, to be shared between all of Menage's virtual processes that utilize the shared storage units; such sharing would be more efficient than creating separate indexes for each of Menage's virtual processes.

373. **Second**, a POSITA would have been motivated to combine Menage with Murphy, because Murphy identifies a known problem (*e.g.*, file management) and provides a solution to address the problem (*e.g.*, creation of an index that contains a listing of the relevant files, the size of the files, the file location and/or other information about the files).

374. Menage similarly describes techniques for file system creation and archival that uses a set of shared storage units that form an initial file system for each virtual process, and a set of private storage units that store changes to the initial file system that occur subsequent to the creation of the file system. A POSITA would have been motivated to incorporate standard, well-known file-management techniques (such as the indexing that is described by Murphy) in the design of a file system such as the one described by Menage. *See* §VII.B.1 above.

375. **Third**, a POSITA would have been motivated to use Murphy's indexing, because it would improve Menage's shared/private storage system. Using Murphy's indexing to improve Menage's system would enable improved file

management, including searching and file location. That is, the combination would merely be a use of a known technique (*e.g.*, indexing) to improve similar methods (*e.g.*, Menage's shared/private storage system) in the same way (*e.g.*, creating an index containing information about the files stored in the system).

376. **Fourth**, a POSITA would have been motivated to combine Menage with Murphy, because the combination would involve applying a known technique (*e.g.*, using Murphy's indexing) to a known device (*e.g.*, Menage's shared/private storage system), which was ready for improvement to yield predictable results. For example, it would be a trivial change to add indexing to Menage's "set of shared storage units," which are located within a "storage device" that is accessible by all virtual processes that utilize the shared storage units. *See* EX1004 (Menage) at 8:8-15 ("[T]he set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101, such as standard application programs, utilities, databases, etc.," and "forms an initial file system 200 for each virtual process 101.").

377. **Fifth**, indexing and its benefits were well-known by 2004. *See* §VII.F above. Adding indexing (as taught by Murphy) to the Menage system would have provided the known benefits of more efficient file searching/locating. Accordingly, modifying the Menage system to use indexing as taught by Murphy is simply the use of a known technique (indexing) to improve similar systems (Menage's

shared/private storage system) to achieve the known benefits of indexing. *See also* §VII.F above.

378. Finally, a POSITA would also have had a reasonable expectation of success in combining Menage with Murphy. Combining Menage with Murphy would merely involve performing indexing in the Menage system. Murphy describes the creation of an "index file" for "a single image file [that] may be created for one or more client computers," which is analogous to Menage's shared storage units. *See* EX1004 (Menage) at 8:10-12 ("In one embodiment, the set of shared storage units 302 is used to store data that is common to the files systems 200 of multiple virtual processes 101.").

2. Claim 14

a. [14.pre] A system for indexing file systems for a plurality of compute nodes, the system comprising:

379. As explained in §XII.A.1.a above with respect to claim limitation [1.pre] in Ground 1, Menage "relates to file system creation and archival, particularly in the context of multiple virtual private servers running on a single physical host machine." EX1004 (Menage) at 3:56-58. As shown in Figure 7 (reproduced and annotated below), "the file system 200 of each virtual process 101 comprises a combination of the shared storage units 302 and the private storage units 304 from one or more templates 310." *Id.*, 8:63-67.

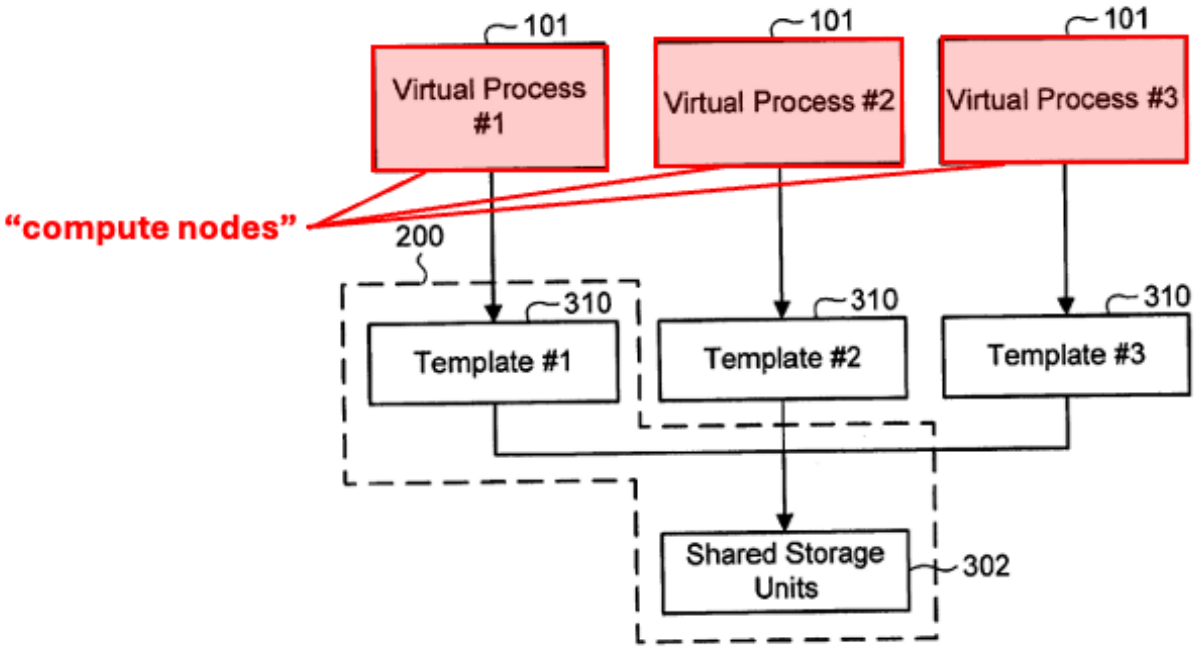


FIG. 7

Menage, Figure 7 (annotated)

380. Murphy discloses indexing: "In various embodiments, a file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may [*sic*] to contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such embodiments, the index file may be created

from the image file, which may be stored on server 206 or on client 202," and "a single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:52-62, 14:67-15:2.

381. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

382. In sum, the combination of Menage and Murphy discloses "a system for indexing file systems [Murphy's index for an image file] for a plurality of compute nodes [Menage's virtual processes 101]."

b. [14.a] a first storage unit configured to store blocks of a root image of said compute nodes;

383. *See* claim limitation [1.a] in Ground 1, in §XII.A.1.b above.

c. [14.b.i] a plurality of second storage units configured to store leaf images of respective compute nodes,

384. *See* claim limitation [1.b.i] in Ground 1, in §XII.A.1.c above.

d. [14.b.ii] said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

385. *See* claim limitation [1.b.ii] in Ground 1, in §XII.A.1.d above.

e. [14.c] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units,

386. *See* claim [4] in Ground 1, in §XII.A.4 above.

f. [14.d] and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

387. Menage and Murphy disclose that at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

388. Specifically, Murphy discloses the indexing of an "image file" (a root image) and the ability to provide the indexing results to another client computer by storing an "index file" on a server: "[A] file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may to [*sic*] contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such

embodiments, the index file may be created from the image file, which may be stored on server 206 or on client 202," and "a single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:52-62, 14:67-15:2.

389. Where a single "image file" is used for multiple client computers, it "may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance." *Id.*, 15:2-5.

390. I note that Murphy's "server 206" is erroneously referred to as "server 202" in the portion quoted in the previous paragraph. This is a typographical error, as the numeral "202" is used consistently throughout Murphy to refer to the "client computers 202," whereas the numeral "206" is used to refer to the "server 206." *See, e.g.*, *Id.*, 3:41-45 ("With reference to FIG. 2, an exemplary management and configuration system 200 suitably includes one or more **client computers 202** … communicating via a network 210 with a **server 206.**"); *id.*, 14:60-62 ("[T]he index file may be created from the image file, which may be stored on **server 206** or on **client 202**.").

391. It would have been obvious to a POSITA to incorporate the indexing taught by Murphy with the Menage system. As explained in §XII.B.2.a above, Menage teaches the use of a "set of shared storage units 302" to store data that is common to multiple virtual processes, and forms "an initial file system." (*i.e.*, "a root image"). A POSITA would have recognized the similarity between Menage's

"shared storage units" and Murphy's "single image" file for use with "multiple client computers." Accordingly, a POSITA would have found it obvious to create an index for Menage's "shared storage units" to be shared between all of the virtual processes that utilize the shared storage units, because such sharing would be more efficient (one shared copy rather than making copies of the index to each computer that needs one).

392. Menage's "set of shared units 302" are located within a "storage device 204" that is accessible by all of the virtual processes that utilize the shared storage units, similar to how Murphy's "server 206" stores the shared image file for access by multiple client computers. Thus, it would have been obvious for a POSITA to store an index for the "set of shared storage units" on the "storage device 204" in order for the index to be provided to or accessed efficiently by the virtual processes.

393. Moreover, a POSITA would have been aware of well-known techniques such as indexing and the sharing of data over networks. *See* §§VII.C and VII.F above.

394. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

395. In sum, the combination of Menage and Murphy discloses that "at least one of said compute nodes [Menage's virtual processes 101] is configured to index said root image [Menage's initial file system 200] and provide the indexing results

[Murphy's index file] to another of said compute nodes [via storage of the index file at Menage's storage device 204 for access by all of the virtual processes 101]."

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

396. *See* claim limitations [14.pre]-[14.d] (§§XII.B.2.a-f above) in this Ground 2, and claim [3] (§XII.A.3 above) in Ground 1.

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices configured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

397. *See* claim limitations [14.pre]-[14.d] ((§§XII.B.2.a-f above) in this Ground 2, and claim [4] (§XII.A.4 above) in Ground 1.

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

398. *See* claim [16] (§XII.B.4 above) in this Ground 2, and claim [5] (§XII.B.5 above) in Ground 1.

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.

399. *See* claim limitations [14.pre]-[14.d] (§§XII.B.2.a-f above) in this Ground 2, and claim [13] (§XII.A.13 above) in Ground 1.

7. Claim 19

a. [19.pre] A method for indexing file systems for a plurality of compute nodes, comprising:

400. *See* claim limitation [14.pre] (§XII.B.2.a above) in this Ground 2.

b. [19.a] storing blocks of a root image of said compute nodes on a first storage unit;

401. *See* claim limitation [14.a] (§XII.B.2.b above) in this Ground 2.

c. [19.b.i] storing leaf images for respective compute nodes on respective second storage units,

402. *See* claim limitation [14.b.i] (§XII.B.2.c above) in this Ground 2.

d. [19.b.ii] said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;

403. *See* claim limitation [14.b.ii] (§XII.B.2.d above) in this Ground 2.

e. [19.c] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;

404. *See* claim limitation [14.c] (§XII.B.2.e above) in this Ground 2.

f. [19.d] receiving indexing results pertaining to said root image from one of said compute nodes; and providing said indexing results to the others of said compute nodes.

405. *See* claim limitation [14.d] (§XII.B.2.f above) in this Ground 2.

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

406. *See* claim limitations [19.pre]-[19.d] (§§XII.B.7.a-f above) in this Ground 2.

407. As explained in §XII.B.2.f above with respect to claim limitation [14.d] in this Ground 2, Murphy discloses the indexing of an "image file" (a root image) and the ability to provide the indexing results to another client computer by storing an "index file" on a server: "[A] file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may [*sic*] to contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such embodiments, the index file may be created from the image file, which may be stored on server 206 or on client 202," and "a single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:52-62, 14:67-15:2.

408. Where a single "image file" is used for multiple client computers, it "may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance." *Id.*, 15:2-5.

409. Thus, it would have been obvious to a POSITA to store an "index file" of Menage's shared storage units 302 within Menage's storage device 204, which, in one embodiment, stores the shared storage units, the private storage units, and the usage maps.

410. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

411. In sum, the combination of Menage and Murphy discloses "storing said indexing results [Murphy's index file] on a shared storage unit [Menage's storage device 204]."

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

412. *See* claim limitations [19.pre]-[19.d] (§§XII.B.7.a-f above) in this Ground 2, and claim [5] (§XII.A.5 above) in Ground 1.

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.

413. *See* claim limitations [19.pre]-[19.d] (§§XII.B.7.a-f above) in this Ground 2, and claim [6] (§XII.A.6 above) in Ground 1.

11. Claim 23

a. [23.pre] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:

414. *See* claim limitation [1.pre] in Ground 1, in §XII.A.1.a above.

b. [23.a] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion,

415. *See* claim limitations [1.a] (§XII.A.1.b above) and [1.b.i] (§XII.A.1.c above) in Ground 1.

c. [23.b] said leaf image portion comprising only additional data blocks not previously contained in said root image portion and, changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node,

416. *See* claim limitation [1.b.ii] in Ground 1, in §XII.A.1.d above.

d. [23.c] wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;

417. *See* claim [4] in Ground 1, in §XII.A.4 above.

e. [23.d] index said root image portion; and provide the results of said indexing to a second compute node,

418. *See* claim limitation [14.d] in this Ground 2, in §XII.B.2.f above.

f. [23.e] wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.

419. As an initial matter, I note that there is no antecedent basis for "said logic encoded in the one or more tangible media" in claim 23. I have been instructed by Petitioner to interpret this phrase as referring to the "instructions" recited in [23.pre] in §XII.B.11.a above.

420. Menage discloses that said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node (virtual process 101): "The present invention relates to file system creation and archival, particularly in the context of multiple virtual private servers running on a single physical host machine." EX1004 (Menage) at 3:56-58. "A virtual private server is an example of a 'virtual process,' which is a set of processes isolated partially or totally from other processes of the system." *Id.*, 3:65-67.

421. It would have been obvious to a POSITA that this limitation describes the general operation of a "virtual private server"—a type of computer—that executes instructions to function. *See* §VII.A above.

422. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

423. In sum, the combination of Menage and Murphy discloses that "said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node [Menage's virtual process 101]."

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.

424. *See* claim limitations [23.pre]-[23.e] (§§XII.B.11.a-f above) and claim [20] (§XII.B.8 above) in this Ground 2.

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

425. *See* claim limitations [23.pre]-[23.e] in this Ground 2, in §§XII.B.11.a-f above.

426. As explained in §XII.B.2.f above with respect to claim limitation [14.d], it would have been obvious to a POSITA to incorporate the indexing taught by Murphy with the Menage system, and to create an index for Menage's "shared storage units" to be shared between all of the virtual processes that utilize the shared storage units.

427. It also would have been obvious to a POSITA to incorporate indexing of each of the individual sets of "private storage units" corresponding to each of Menage's virtual processes, such that any changes (or additions) to the "initial file system" in the set of shared storage units would also be indexed, since it is the

combination of the shared and private storage units that form the complete file system.

428. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

429. In sum, the combination of Menage and Murphy discloses that "the instructions further cause the at least one computing device [Menage's virtual process 101] to index [Murphy's index file] said leaf image portion [changes to the Menage's initial file system 200 that occur subsequent to the creation of the file system 200]."

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

430. *See* claim [25] in this Ground 2, in §XII.B.13 above.

431. It would have been obvious to a POSITA to re-index the sets of private storage units periodically as they are modified by their respective virtual processes to ensure that the index is up-to-date and includes all modifications and additions to the initial file system stored in the shared storage units. Furthermore, it would have been obvious to a POSITA that, because the complete file system for each virtual process is the combination of the "initial file system" stored in the shared storage units along with the changes made to the initial file system that are stored in the set

of private storage unit, that the indexes of the private storage units and shared storage units would need to be combined (*i.e.*, "merged" as required by claim [26]) in order to have a complete index for the full file system. *See also* §VII.F above.

432. A POSITA would have been motivated to combine Menage and Murphy for the reasons discussed in §XII.B.1 above.

433. In sum, the combination of Menage and Murphy discloses that "the instructions further cause the at least one computing device [Menage's virtual process 101] to re-index [Murphy's index file] said file system by re-indexing said leaf image portion [changes to the Menage's initial file system 200 that occur subsequent to the creation of the file system 200] and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion [Menage's initial file system 200]."

15. Claim 27: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

434. *See* claim limitations [23.pre]-[23.e] (§§XII.B.11.a-f above) in this Ground 2, and claim [6] (§XII.A.6 above) in Ground 1.

C. Ground 3: Birse in View of Rothman Renders Claims 1-13 Obvious

1. Motivation to Combine Birse with Rothman

435. A POSITA would have been motivated to combine Birse and Rothman for at least the following five reasons.

436. **First**, the combination of Birse and Rothman represents the use of known techniques according to known methods in the same way to yield predictable results. Birse and Rothman both describe systems wherein clients are booted by accessing a boot image over a network. For example, in Birse, the NC clients "boot over the network by accessing a stored copy of the operating system from an NC server." Likewise, in Rothman, "the client obtains an operating system boot image over the network from a boot server."

437. The Birse and Rothman combination would merely involve using Rothman's caching in Birse's system, both of which were known, according to Rothman's methods for caching a boot image. For example, Birse teaches the use of a "system volumes" to store data that is common to multiple NC clients. *See* EX1006 (Birse) at 14:27-34, 14:41-44 ("[T]he NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 … The core system volume 922 preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core system volume 922 is to provide all the system components that are mandatory for system

operation…. The user system volume 924 contains all the user-configurable system components, including preferences and all the system additions installed by application software, such as application-installed extensions and libraries.").

438. Rothman similarly describes the caching of a "boot image" that is "executed to boot the operating system." *See* EX1007 (Rothman) at 4:6-12 ("In a block 204, the client obtains an operating system boot image over the network from a boot server. In conjunction with delivering the boot image to the client, the boot image is cached at a network device, such as a switch, router, bridge, etc., or a gateway server that is proximate to the client in a block 206. The image is then booted in a block 208 to enable run-time use, which is continued in a continuation block 210.").

439. A POSITA would have recognized the similarity between Birse's "system volumes" and Rothman's "boot image," and therefore would have found it obvious to add caching (as taught by Rothman) for the "system volumes" in Birse's system, to provide for faster access to frequently retrieved data.

440. **Second**, a POSITA would have been motivated to combine Birse with Rothman, because Rothman identifies a known problem (*e.g.*, network boot efficiency) and provides a solution to address the problem (*e.g.*, storage of a boot image in a cache for faster access by a client computer).

441. Birse describes techniques for a file system that uses a set of shared system volumes and one or more shadow volumes for each NC client that store modifications that are made to the system volumes. A POSITA would have been motivated to incorporate standard, well-known file-management techniques (such as the caching that is described by Rothman) in the design of a file system such as the one described by Birse.

442. Indeed, Birse recognizes that a "cache" may be included in an NC client or NC server: "FIG. 2 is a block diagram of a digital processing system which may be used in accordance with one embodiment of the present invention. For example, the digital processing system 200 shown in FIG. 2 may be used as an NC client, an NC server, or other server computer system, such as an NT server. … The digital processing system 200 includes a processor 252, which may represent one or more processors and may include one or more conventional types of such processors, such as Motorola PowerPC processor, an Intel Pentium (or x86) processor, etc. … The processor may also be coupled to other types of storage areas/memories (e.g., **cache**, Flash memory, disk, etc.), which could be considered as part of the memory 254 or separate from the memory 254." EX1006 (Birse) at 7:63-8:1, 8:10-20.

443. **Third**, a POSITA would have been motivated to use Rothman's caching, because it would improve Birse's system/shadow volume storage system. Using Rothman's caching to improve Birse's system would enable faster access to

frequently accessed files retrieved by the NC clients from the system volumes. That is, the combination would merely be a use of a known technique (*e.g.*, caching) to improve similar methods (*e.g.*, Birse's system/shadow volumes) in the same way (*e.g.*, caching data retrieved from the system volumes by an NC client).

444. **Fourth**, a POSITA would have been motivated to combine Birse with Rothman, because the combination would involve applying a known technique (*e.g.*, using Rothman's caching) to a known device (*e.g.*, Birse's system/shadow volumes), which was ready for improvement to yield predictable results. For example, it would be a trivial change to add caching to Birse's system volumes, which are located at an "NC server 170" that is accessible by each of the NC clients. *See* EX1006 (Birse) at 5:10-11 ("The NC server contains one or more system volumes.").

445. **Fifth**, caching was already a ubiquitous technique used in computer storage long before 2004. *See* §VII.D above. Adding caching (as taught by Rothman) to the Birse system would have provided the known benefit of faster access to frequently used data. Accordingly, modifying the Birse system to use caching as taught by Rothman is simply the use of a known technique (caching) to improve similar systems (Birse's NC client/server system) to achieve the known benefits of caching.

446. Finally, a POSITA would also have had a reasonable expectation of success in combining Birse with Rothman. Combining Birse with Rothman would

merely involve performing caching in the Birse system. Rothman describes the caching of a "boot image," which is analogous to Birse's system volumes. *See* EX1006 (Birse) at 5:10-13 ("The system volumes 174 include a protected, read-only, master copy of the operating system software.").

2. Claim 1

a. [1.pre] A system for providing data to a plurality of compute nodes, the system comprising:

447. Birse teaches a system for providing data (from NC server 170) to a plurality of compute nodes (NC clients): "… [T]he NC clients [150] boot over the network by accessing a stored copy of the operating system from the NC server [170]. Additionally, when NC clients want to run an application, they access it from the NC server." EX1006 (Birse) at 4:56-60, Figure 1 (reproduced and annotated below). *See also id.*, 2:51-3:14 ("A method and apparatus are described for providing a reliable and maintainable operating system in a net-booted environment. … According to another embodiment, a network administrator can upgrade every NC client in the NC system to a new version of the operating system by simply replacing a single file on the NC server. Further, according to another feature of the present invention, the network administrator can perform such an upgrade remotely from any NC client of the NC system. Advantageously, in this manner, NC system administration and maintenance costs are kept low as compared

to a typical network of thick clients that each has a local copy of the operating system that must be replaced.").

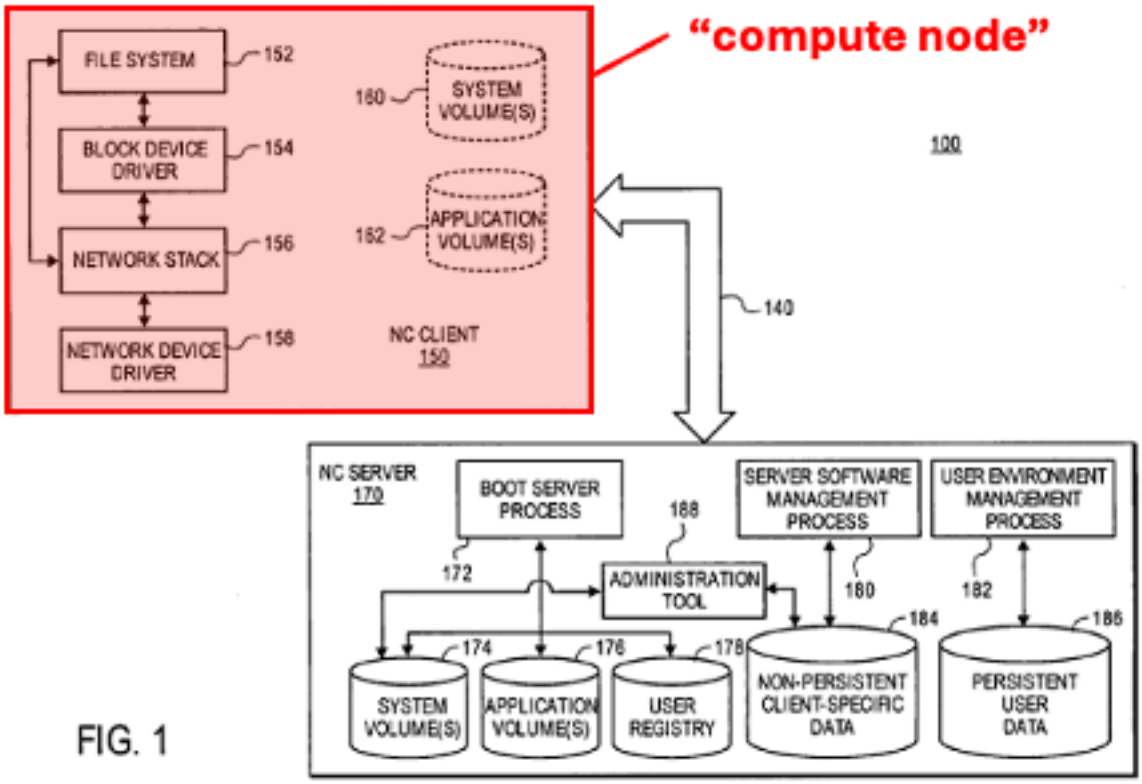


FIG. 1

Birse, Figure 1 (annotated)

448. "FIG. 8 is a simplified block diagram illustrating two NC systems 810 and 820 coupled via the Internet 805." *Id.*, 14:8-9, Figure 8 (reproduced and annotated below).

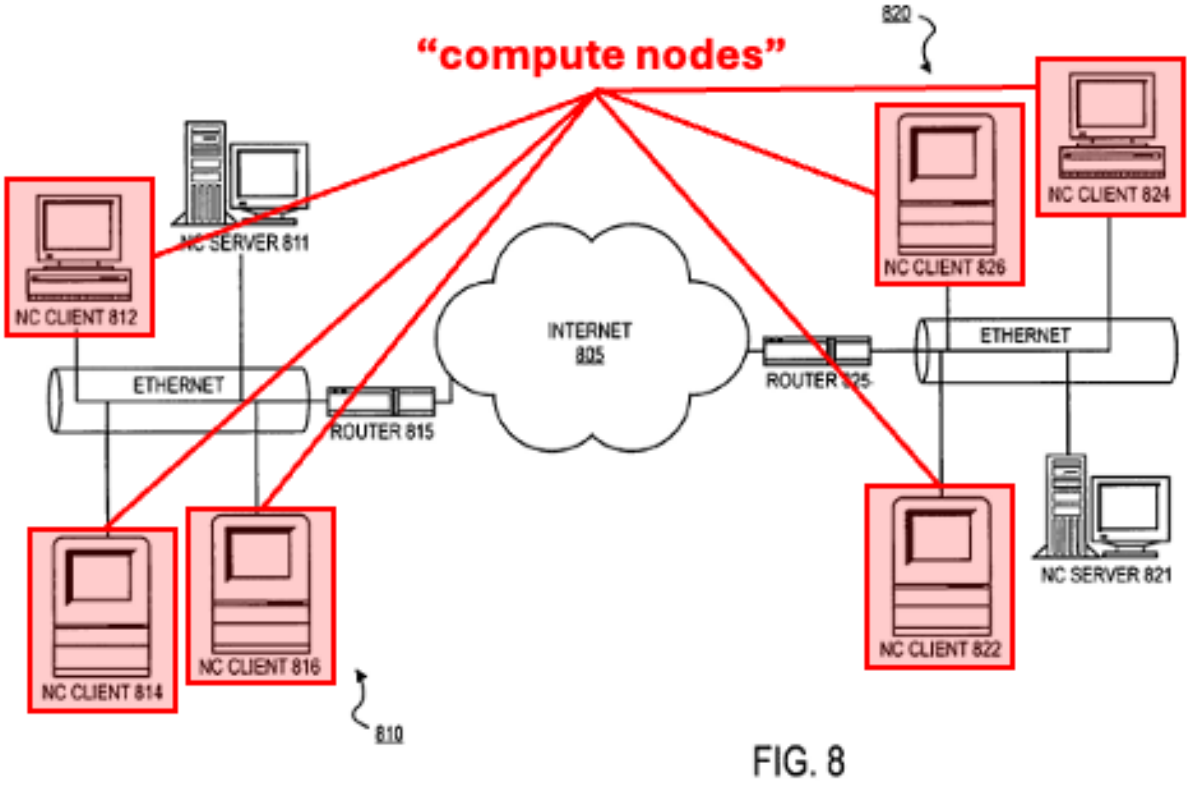


FIG. 8

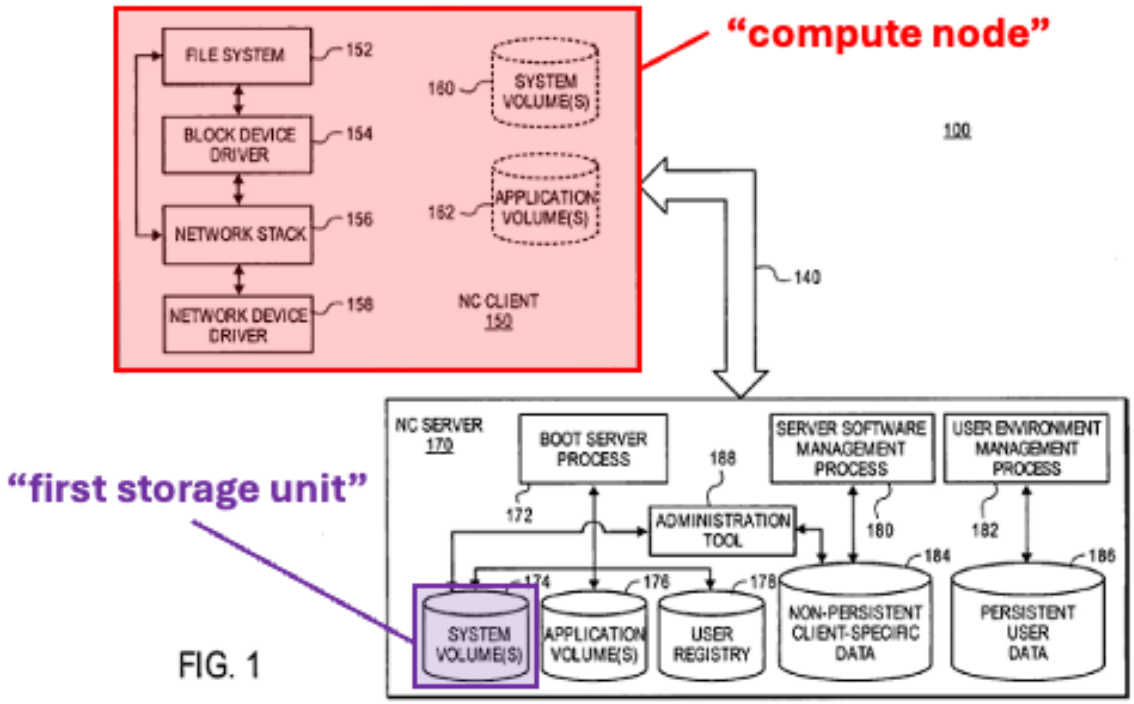
Birse, Figure 8 (annotated)

449. In sum, Birse discloses "A system for providing data [a stored copy of the operating system at NC server 170] to a plurality of compute nodes [NC clients 150]."

b. [1.a] a first storage unit configured to store blocks of a root image of said compute nodes;

450. Birse discloses storing blocks of a root image (system volumes 174, 660, 922, 924) of said compute nodes (multiple NC clients 150) on a first storage unit (one or more hard drives 610, logically separate in NC folders/directories 630, 640): "The NC server 170 includes one or more system volumes 174…. The system

volumes 174 include a protected, read-only, master copy of the operating system software." EX1006 (Birse) at 5:10-14, Figure 1 (reproduced and annotated below).



Birse, Figure 1 (annotated)

451. According to Birse, "the NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 … The core system volume 922 preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core system volume 922 is to provide all the system components that are mandatory for system operation…. The user system volume 924 contains all the user-configurable system components, including preferences and all the system additions installed by

application software, such as application-installed extensions and libraries." *Id.*, 14:27-34, 14:41-44.

452. Birse discloses that the file 660, storing the shared OS image (within the folder 640), is a first storage unit storing blocks of a root image. "The shared operating system image 660 is a read-write version of the read-only master operating system image 650 and may be mounted by the NC client 150 as part of step 395." *Id.*, 12:44–47; *see also id.*, 7:6-10 ("The NC client also includes … a block device driver."), 7:35-36 ("The block device driver 154 services file system read and write requests to the system volumes 160.").

453. The shared system volume 660 (a.k.a. system volumes 174, 922, 924, 1010) is stored as blocks as shown in Fig. 10 (reproduced below):

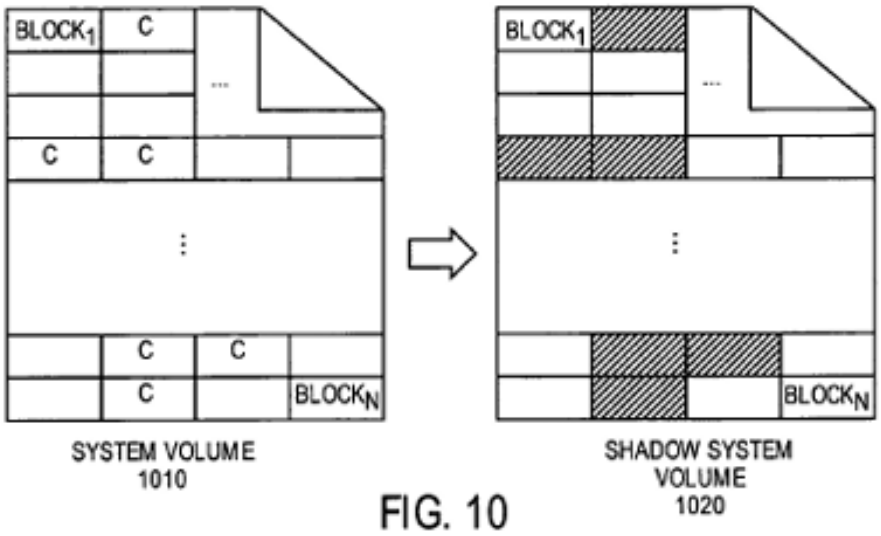


FIG. 10

Birse, Figure 10

454. In sum, Birse discloses "a first storage unit [one or more hard drives 610, logically separate in NC folders/directories 630, 640] configured to store blocks of a root image [system volumes 174, 660, 922, 924] of said compute nodes [NC clients 150]."

c. [1.b.i] a plurality of second storage units configured to store leaf images of respective compute nodes,

455. Birse discloses storing leaf images (shadow volumes/images) for respective compute nodes (NC clients) on respective second storage units (NC client folders): "[I]t is assumed that the shadow volume contains only those portions of the operating system that have been modified by the user rather than a complete copy of the operating system as modified." EX1006 (Birse) at 11:49-52, Figure 9 (reproduced and annotated below).

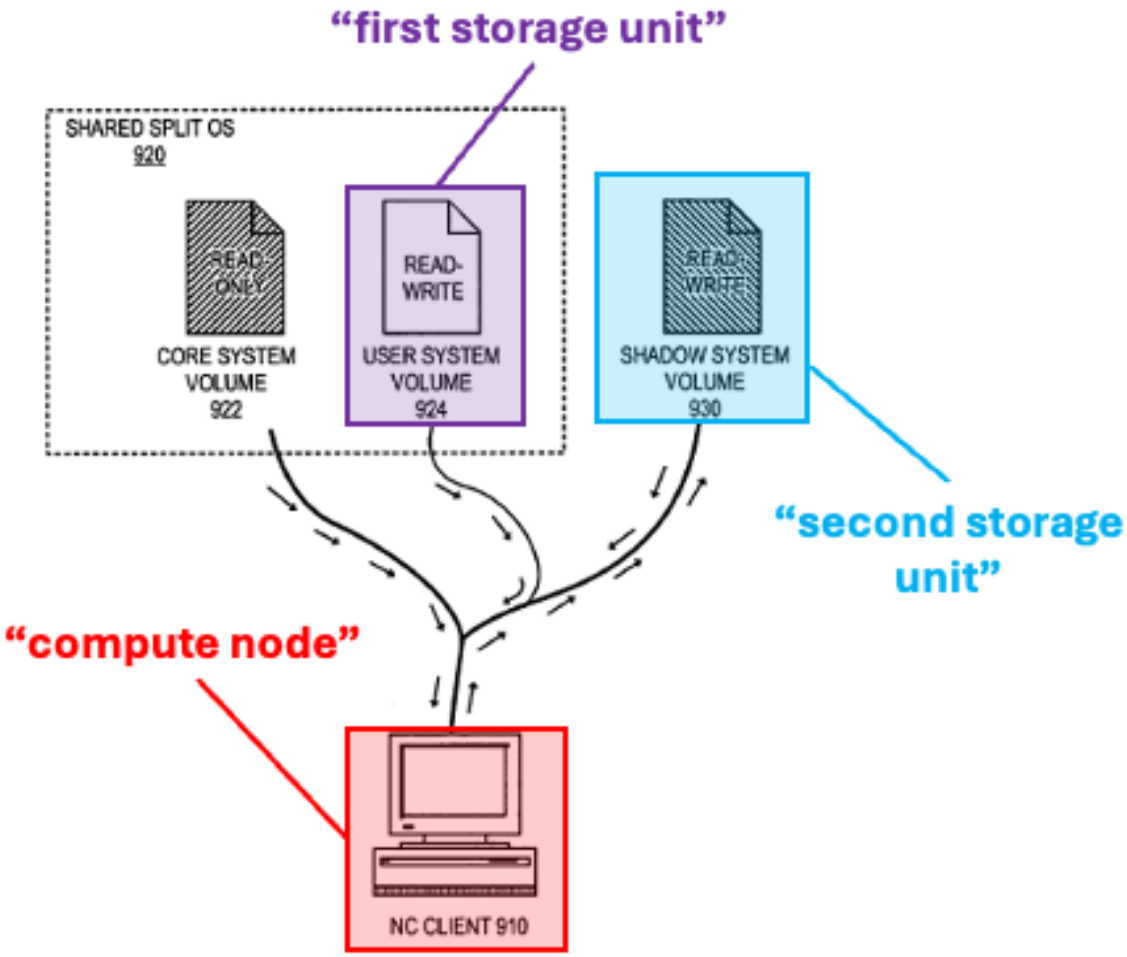


FIG. 9

Birse, Figure 9 (annotated)

456. According to Birse, "the NC server 170 also creates a shadow system volume 930 for each connected NC client. The shadow system volume 930 shadows the user system volume 924 by storing modifications that are made to the user system volume 924. In alternative embodiments, the NC server 170 may provide a separate user system volume 924 for each connected NC client." *Id.*, 14:49-55.

457. In sum, Birse discloses "a plurality of second storage units [shadow system volumes 930] configured to store leaf images [modifications that are made to the user system volume 924] of respective compute nodes [NC clients 150]."

d. [1.b.ii] said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

458. Birse discloses leaf images including only additional data blocks not previously contained in said root image (system volumes) and changes made by respective compute nodes to the blocks of said root image (write requests for new data and changes to operating system data), wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes (only new data and changes are written): "[O]ne or more shadow volumes, may be created for the NC client 150 and stored with the non-persistent client-specific data 184 for the purpose of storing user preferences and/or changes to the operating system." EX1006 (Birse) at 7:42-46. "[T]he shadow volume contains only those portions of the operating system that have been modified by the user rather than a complete copy of the operating system as modified." *Id.*, 11:49-52.

459. Birse's Figure 4 "is a flow diagram illustrating how NC client [150] write requests directed to the system volumes 160 are handled… [A]t step 410, the file system 152 generates a write request directed to the system volumes 160. The write request is received by the block device driver 154 and redirected by the block device driver 154 at step 420 by translating it into a write request directed to the user's shadow volume on the NC server 170. At step 430, the NC server 170 stores information associated with the write request in the shadow volume associated with the NC client 150. In this manner, the file system 152 need not be aware that attempted modifications to the system volumes 160 are recorded instead in a client-specific shadow volume residing on the NC server 170." *Id.*, 10:63-65, 11:6-17, Figure 4 (reproduced below).

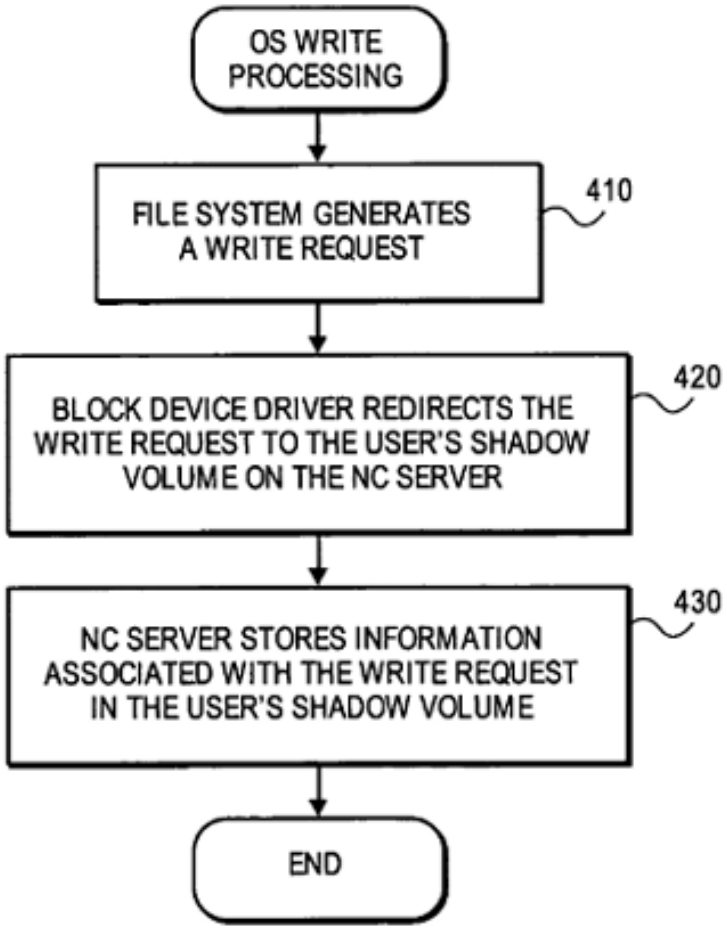


FIG. 4

Birse, Figure 4

460. Birse's Figure 5 "is a flow diagram illustrating how NC client read requests directed to the system volume are handled." *Id.*, 2:21-23. Birse explains that "at step 510, the file system 152 generates a read request directed to the system volumes 160. At step 520, the read request is received by the block device driver

154 and a determination is made whether the read request specifies a portion of the operating system that has been modified by the user or whether the read request specifies a portion of the operating system that remains unchanged. If the read request corresponds to a portion of the operating system that has not been modified by the user, the processing continues with step 530. However, if the read request corresponds to a portion of the operating system that has been previously modified by the user, then processing continues with step 540." *Id.*, 11:55-67, Figure 5 (reproduced below).

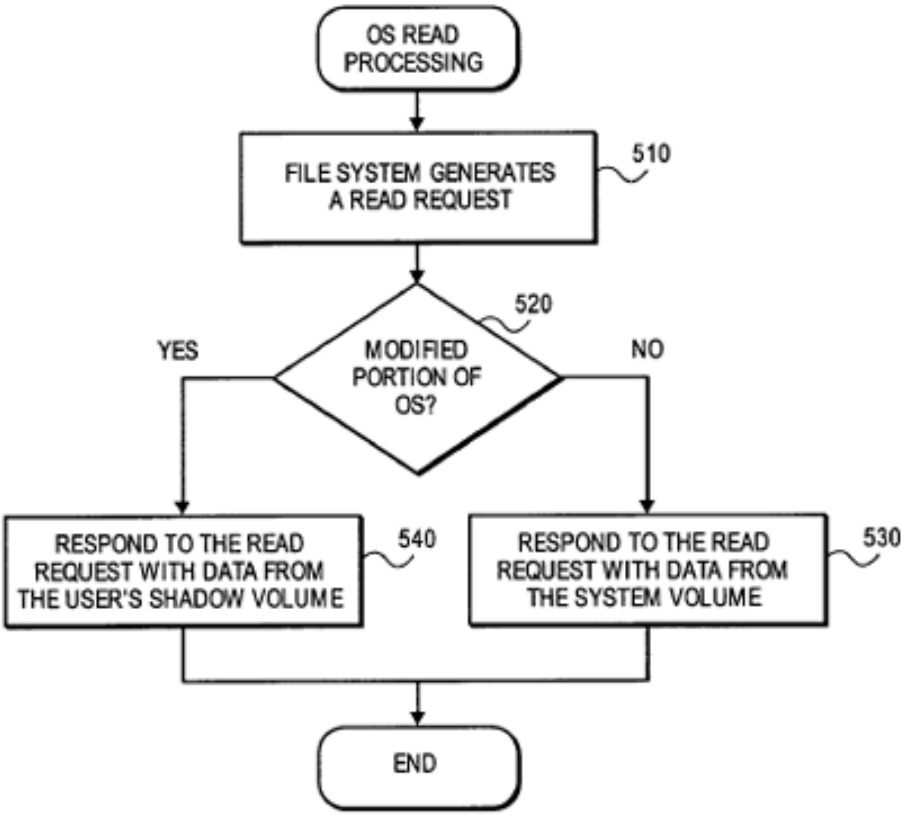


FIG. 5

Birse, Figure 5

461. "At step 530, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the system volumes 174." *Id.*, 12:1-3.

462. "At step 540, the block device driver directs the NC server 170 to retrieve the information associated with the read request from the user's shadow volume." *Id.*, 12:4-6.

463. Thus, Birse teaches that "the file system 152 need not be aware of the particular mechanism used to coordinate user modifications with the original operating system as contained in the system volumes 174." *Id.*, 12:7-10.

464. As explained in §XII.C.2.c above with respect to claim limitation [1.b.i], "the NC server "creates a shadow system volume 930 for each connected NC client" that "shadows the user system volume 924 by storing modifications that are made to the user system volume." *Id.*, 14:49-55, Figure 9 (reproduced and annotated below).

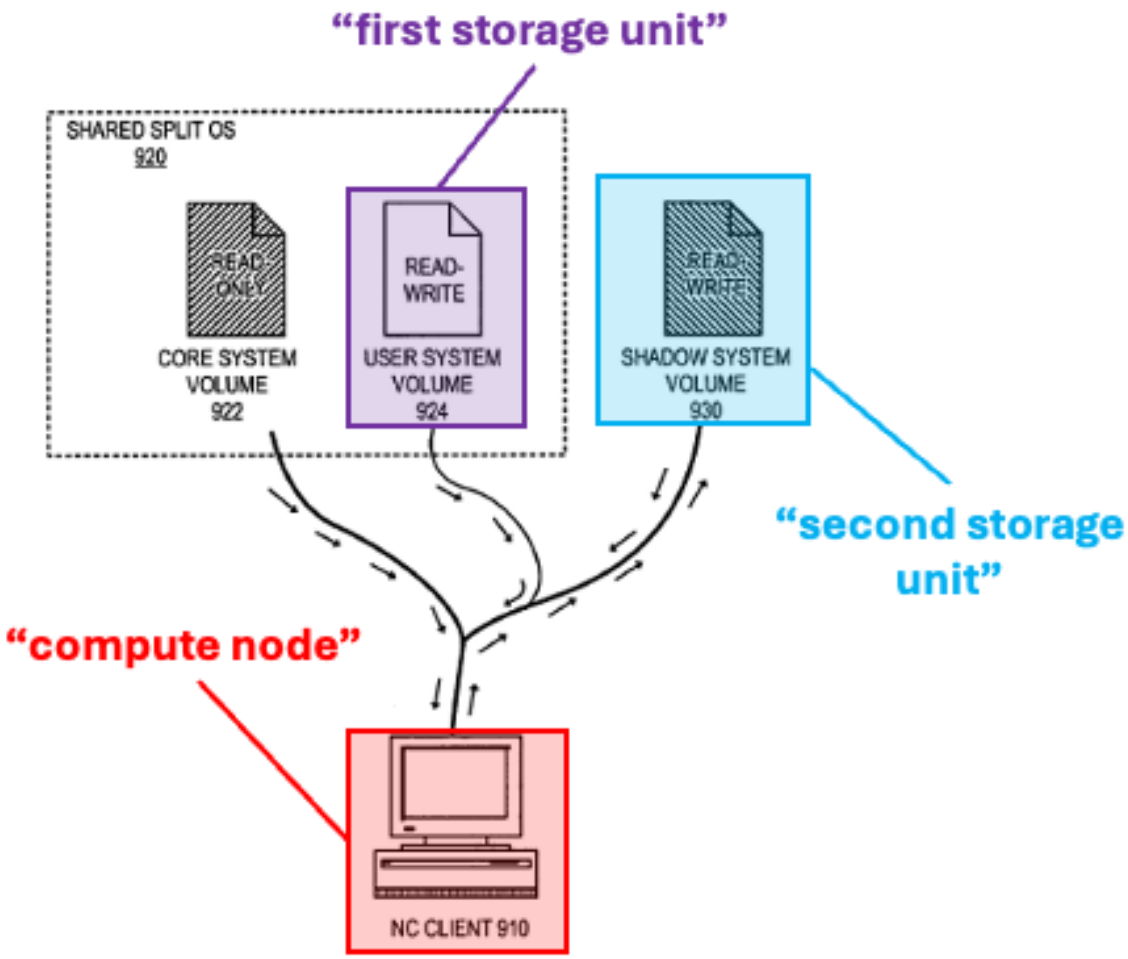


FIG. 9

Birse, Figure 9 (annotated)

465. In the embodiment shown in Figure 9, "[t]he shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) directed to the user system volume 930 are instead copied to the shadow system volume 930," and "the shadow system volume 930 is not visible to the user." *Id.*, 14:56-62. As indicated by the

directional arrows in Figure 9, "data may be read from each of the core system volume 922, the user system volume 924, and the shadow system volume 930; however, data may only be written to the shadow system volume 930." *Id.*, 15:1-7.

466. In sum, Birse discloses "said leaf images including only additional data blocks not previously contained in said root image [system volumes 922, 924] and changes made by respective compute nodes to the blocks of said root image [modifications that are made to the user system volume 924], wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes."

e. [1.c] a cache configured to cache blocks of said root image previously accessed by at least one of said compute nodes.

467. The combination of Birse and Rothman discloses a cache configured to cache blocks of said root image (system volumes) previously accessed by at least one of said compute nodes (NC clients).

468. Birse teaches that an NC client or an NC server may include a "cache" memory and a "cache bus": "FIG. 2 is a block diagram of a digital processing system which may be used in accordance with one embodiment of the present invention. For example, the digital processing system 200 shown in FIG. 2 may be used as an NC client, an NC server, or other server computer system, such as an NT server. … The digital processing system 200 includes a processor 252, which may represent

one or more processors and may include one or more conventional types of such processors, such as Motorola PowerPC processor, an Intel Pentium (or x86) processor, etc. … The processor may also be coupled to other types of storage areas/memories (e.g., **cache**, Flash memory, disk, etc.), which could be considered as part of the memory 254 or separate from the memory 254." EX1006 (Birse) at 7:63-8:1, 8:10-20.

469. Rothman similarly describes a system where "boot image file(s)" and "boot program bootstrap files" are cached at a network device or gateway server proximate to a client that is configured to boot using a "network boot process": "In a block 204, the client obtains an operating system boot image over the network from a boot server. In conjunction with delivering the boot image to the client, **the boot image is cached** at a network device, such as a switch, router, bridge, etc., or a gateway server that is proximate to the client in a block 206. The image is then booted in a block 208 to enable run-time use, which is continued in a continuation block 210." EX1007 (Rothman) at 4:6-12, Figure 2 (reproduced below).

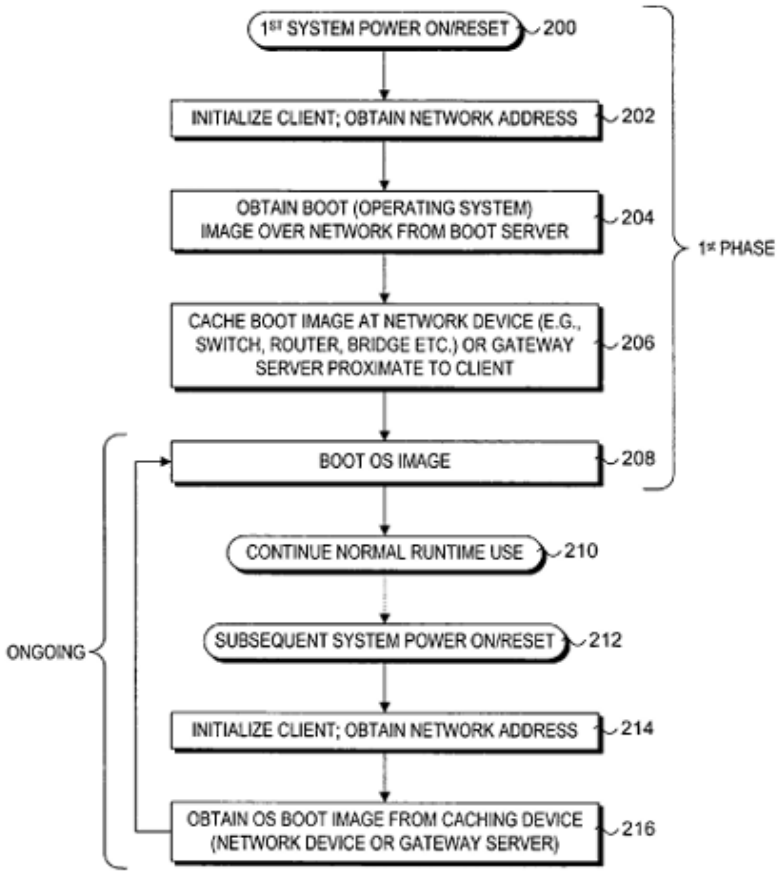


Fig. 2

Rothman, Figure 2

470. Furthermore, in Rothman, "[a]t continuation block 212, a subsequent system power on or reset event occurs. … As before, the client is initialized, enabling the client to obtain a network address in a block 214. This time, however, the client obtains the OS boot image from the **caching device** (e.g., switch, router, bridge, gateway server, etc.) rather than the boot server." *Id.*, 4:13-22.

471. Rothman's Figure 1*a* "depicts the first phase of one embodiment of the invention, wherein a network bootstrap program file and **a boot image are cached**

on a network device in conjunction with downloading the same from a boot server to a client." *Id.*, 2:1-5, Figure 1*a* (reproduced and annotated below).

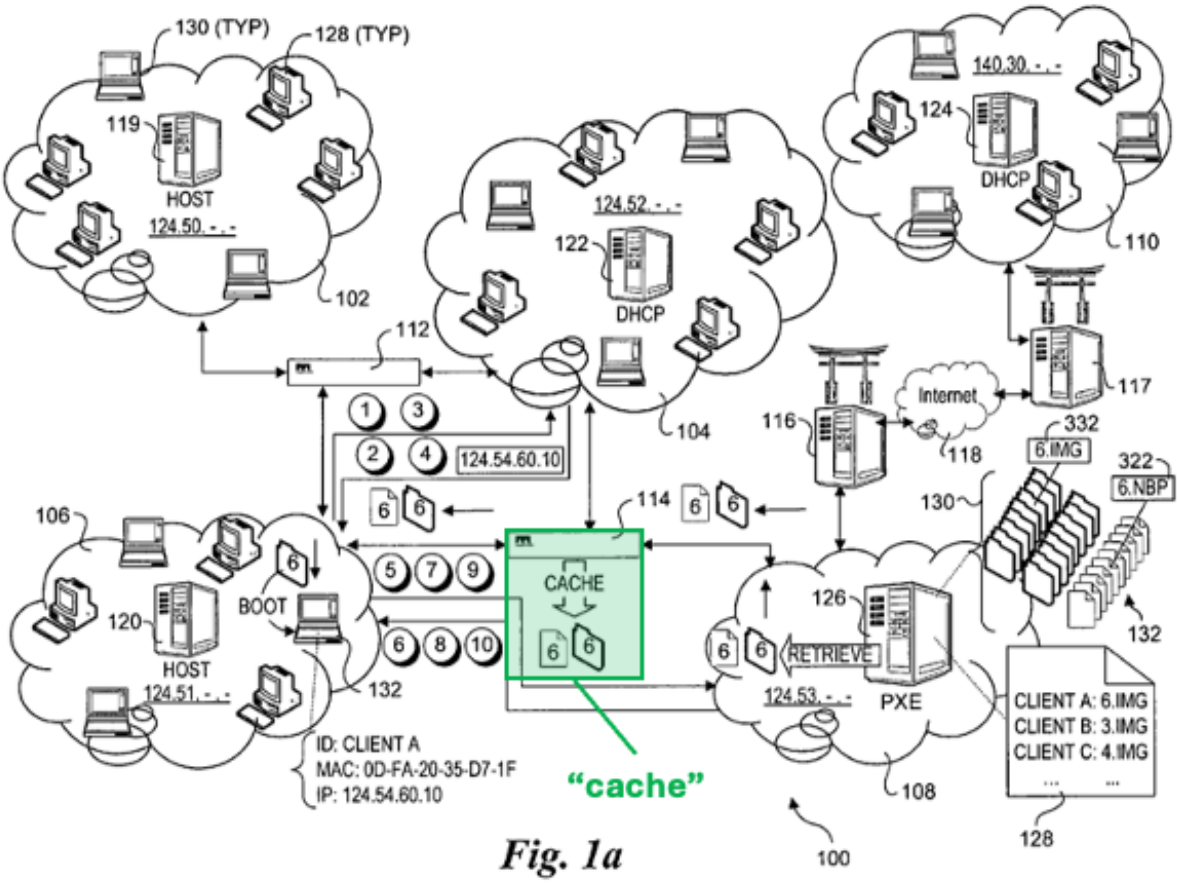


Fig. 1a

Rothman, Figure 1*a* (annotated)

472. Rothman's Figure 1*b* "depicts a second phase of the download technique, wherein **the cache boot image is downloaded directly from the network device**." *Id.*, 2:6-8, Figure 2*a* (reproduced and annotated below); *see also id.*, 8:5-6 ("In the example of FIG. 1*a*, **the caching device is switch 114**."); *id.*, 9:22-26 ("In response to receiving the boot image download request message, one or more corresponding **boot image files, as depicted by boot image 6 in FIG. 1*b*, is/are**

retrieved from a local storage device hosted by switch 114 and downloaded to client 132 at step 10.").

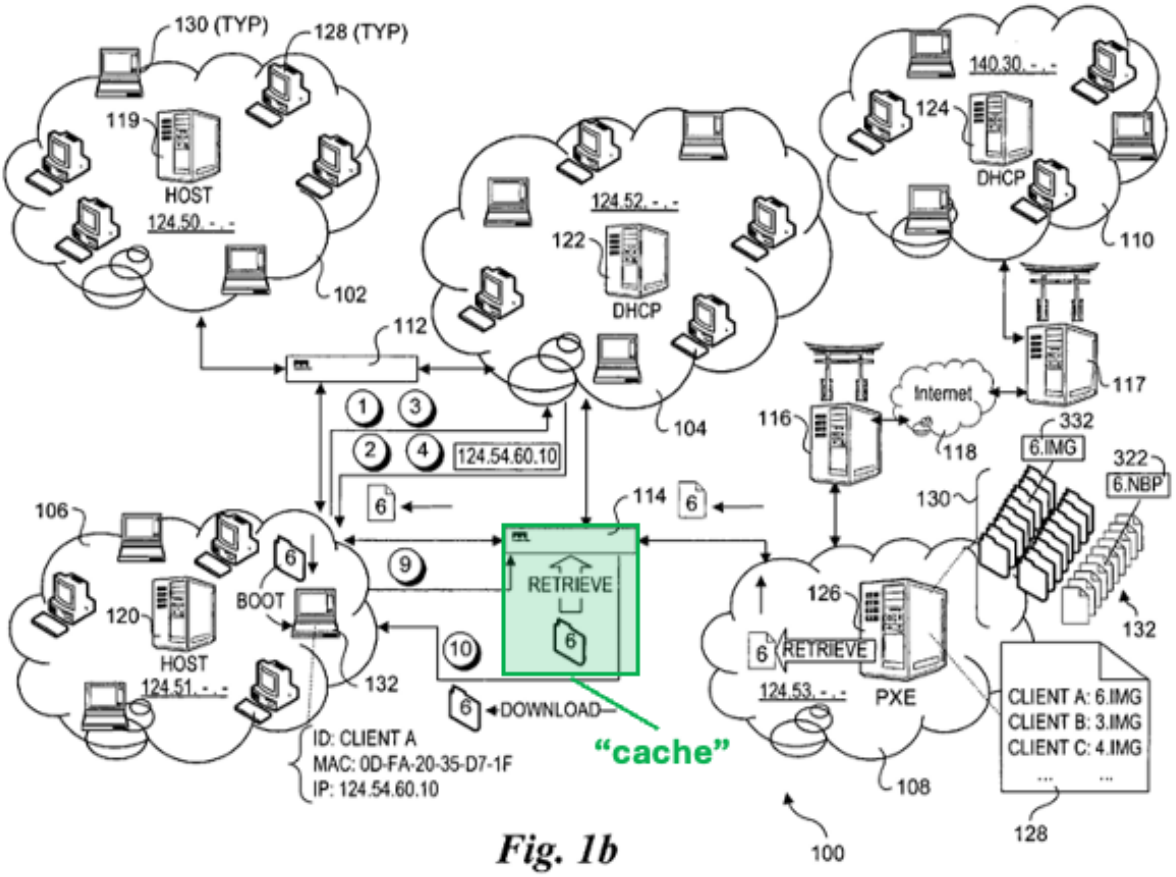


Fig. 1b

Rothman, Figure 1*b* (annotated)

473. It would have been obvious to a POSITA to combine the caching techniques taught by Rothman with Birse's system to improve the performance of read requests directed to Birse's system volumes, as further detailed in §XII.C.1 above. In such a modified system, a cache memory would have been used to cache blocks of data requested directly from one of the system volumes upon access by one of the NC clients. On a subsequent read request for the same data (from the NC

client that originally requested the data, or any other NC client requesting the same data from the shared volumes), the data would be retrieved by the block device driver from the faster cache memory instead of the system volume. A POSITA would have also understood that caching is a well-known technique heavily taught and used. *See also* §VII.D above.

474. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

475. In sum, the combination of Birse and Rothman discloses "a cache [Rothman's caching device] configured to cache blocks of said root image [Birse's system volumes 922, 924] previously accessed by at least one of said compute nodes [Birse's NC clients 150]."

3. Claim 2: The system as recited in claim 1 wherein said cache is configured to store X most recently accessed blocks of said root image, and wherein X represents a cache threshold value.

476. *See* claim limitations [1.pre]-[1.c] in this Ground 3, in §§XII.C.2.a-e above.

477. A POSITA would have understood that this limitation describes nothing more than common caching behavior (cache management policies, where caches naturally have limited room and must use thresholds to decide what to keep/evict).

478. Specifically, this limitation describes the "least-recently used" or "LRU" cache management policy. *See* §VII.D.1 above. As explained above, LRU is one of the oldest, most effective, and best-known algorithms. Thus, this limitation would have been obvious to a POSITA for the same reasons discussed above regarding Menage and the knowledge of a POSITA. *See* claim [2] in Ground 1, in §XII.A.2 above.

479. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

480. In sum, the combination of Birse and Rothman discloses that "said cache [Rothman's caching device] is configured to store X most recently accessed blocks of said root image [Birse's system volumes 922, 924], and wherein X represents a cache threshold value."

4. Claim 3: The system as recited in claim 1 wherein said first storage unit, said second storage units, and said cache are contained within a single storage appliance.

481. *See* claim limitations [1.pre]-[1.c] in this Ground 3, in §§XII.C.2.a-e above.

482. The combination of Birse and Rothman teaches that said first storage unit (shared system volumes), said second storage units (shadow volumes), and said cache are contained within a single storage appliance (NC server).

483. As explained in §XII.C.2.e above with respect to claim limitation [1.c], Birse describes the inclusion of a "cache" in the NC server. *See* EX1006 (Birse) at 7:63-8:1, 8:10-20. Birse also discloses embodiments where the system volumes and the shadow volumes are stored together at an NC server. *See, e.g.*, *id.*, 12:30-63, Figure 6 (showing a directory structure for an NC server that includes a shared operating system image and shadow images for one or more NC clients).

484. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

485. In sum, the combination of Birse and Rothman discloses that "said first storage unit [Birse's one or more hard drives 610, logically separate in NC folders/directories 630, 640], said second storage units [Birse's shadow system volumes 930], and said cache [Rothman's caching device] are contained within a single storage appliance [Birse's NC server 170]."

5. Claim 4: The system as recited in claim 1 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit, respective second storage units, and said cache to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

486. *See* claim limitations [1.pre]-[1.c] in this Ground 3, in §§XII.C.2.a-e above.

487. Birse discloses a plurality of union block devices (block device drivers) configured to interface between respective compute nodes (NC clients) and said first storage unit (system volumes), respective second storage units (shadow volumes), and said cache to distribute application environments by merging the blocks of said root image (core system volume image) with the blocks of respective leaf images (portions of the operating system modified by the user): "[T]he NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 … The core system volume 922 preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core system volume 922 is to provide all the system components that are mandatory for system operation." EX1006 (Birse) at 14:27-34.

488. Furthermore, "the NC server 170 also creates a shadow system volume 930 for each connected NC client. The shadow system volume 930 shadows the user system volume 924 by storing modifications that are made to the user system volume 924. In alternative embodiments, the NC server 170 may provide a separate user system volume 924 for each connected NC client" and "the shadow system volume 930 is used by the block device driver 154 of the NC client 150 to implement a 'copy-on-write' storage scheme in which modifications (writes) directed to the user system volume 930 are instead copied to the shadow system volume 930." *Id.*, 14:49-61.

489. The Birse system (modified in light of Rothman to include a cache) operates the same way as the '844 patent's description of a "branching store file system" to provide an "application environment." Specifically, the system volumes include "a protected, read-only, master copy of the operating system software," and a "shadow volume contains only those portions of the operating system that have been modified by the user." *Id.*, 5:13-14, 11:49-52. The block device driver redirects read and write requests between the system volumes and shadow volumes in a way that is transparent to the file system—in one embodiment, "the block device driver 154 appears to the file system 152 to be a standard hard drive device driver," and "the file system 152 need not be aware of the particular mechanism used to coordinate user modifications with the original operating system as contained in the system volumes." *Id.*, 7:35-38, 11:14-17.

490. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

491. In sum, the combination of Birse and Rothman discloses "a plurality of union block devices [Birse's block device driver(s) 154] configured to interface between respective compute nodes [Birse's NC clients 150] and said first storage unit [Birse's one or more hard drives 610, logically separate in NC folders/directories 630, 640], respective second storage units [Birse's shadow system volumes 930], and said cache [Rothman's caching device] to distribute

application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image [Birse's system volumes 922, 924] with the blocks of respective leaf images [modifications that are made to Birse's user system volume 924]."

6. Claim 5: The system as recited in claim 4 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

492. *See* claim [4] in this Ground 3, in §XII.C.5 above.

493. As explained in §XII.C.2.d above with respect to claim limitation [1.b.ii] and claim [4], Birse teaches a plurality of union block devices (block device drivers) that "keep track of which portions of the operating system that have been modified by the user," and directs read and write requests between the system volumes and shadow volumes as necessary. EX1006 (Birse) at 12:11-15. Birse also indicates that "[t]he block device driver 154 services file system read and write requests to the system volumes 160" and "[i]n one embodiment, the block device driver 154 appears to the file system 152 to be a standard hard drive device driver." *Id.*, 7:35-38. A POSITA would have understood that a hard drive device driver logically resides in the lower-level layers of the operating systems. *See also* §VII.A above.

494. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

495. In sum, the combination of Birse and Rothman discloses that "said union block devices [Birse's block device driver(s) 154] comprise low-level drivers [Birse's hard drive device drivers] for interfacing between the file systems of respective compute nodes [Birse's NC clients 150] and said first storage unit [Birse's one or more hard drives 610, logically separate in NC folders/directories 630, 640], respective second storage units [Birse's system volumes 922, 924], and said cache [Rothman's caching device]."

7. Claim 6: The system as recited in claim 1 wherein said first storage unit is read-only.

496. *See* claim limitations [1.pre]-[1.c] in this Ground 3, in §§XII.C.2.a-e above.

497. As explained in §XII.C.2.d above with respect to claim limitation [1.b.ii], Birse teaches that write requests to a system volume are redirected by the block device driver by translation into "a write request directed to the user's shadow volume on the NC server." EX1006 (Birse) at 11:8-11. Thus, it would have been obvious to a POSITA that because all write requests from an NC client that are directed to a system volume are redirected to the user's shadow volume, the system volume (*i.e.*, the "first storage unit") is read-only.

498. In sum, Birse discloses that "said first storage unit [Birse's one or more hard drives 610, logically separate in NC folders/directories 630, 640] is read-only."

8. Claim 7

a. [7.pre] A method for providing data to a plurality of compute nodes, comprising:

499. *See* claim limitation [1.pre] in this Ground 3, in §XII.C.2.a above.

b. [7.a] storing blocks of a root image of said compute nodes on a first storage unit;

500. *See* claim limitation [1.a] in this Ground 3, in §XII.C.2.b above.

c. [7.b.i] storing leaf images for respective compute nodes on respective second storage units,

501. *See* claim limitation [1.b.i] in this Ground 3, in §XII.C.2.c above.

d. [7.b.ii] said leaf images including only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

502. *See* claim limitation [1.b.ii] in this Ground 3, in §XII.C.2.d above.

e. [7.c] caching blocks of said root image that have been accessed by at least one of said compute nodes in a cache memory.

503. *See* claim limitation [1.c] in this Ground 1, in §XII.C.2.e above.

9. Claim 8: The method as recited in claim 7 further comprising: receiving a read request from at least one of said compute nodes, wherein a first portion of the data requested is currently stored in said cache memory; and providing said first portion of said data to said at least one of said compute nodes from said cache memory.

504. *See* claim limitations [7.a]-[7.c] in this Ground 3, in §§XII.C.8.b-e above.

505. As explained with respect to claim limitation [1.c] in §XII.C.2.e above, Rothman teaches the use of a cache: "the boot image file(s) is/are cached at the network device or gateway server proximate to the client." EX1007 (Rothman) at 8:56-58. Caching the boot image at a location "proximate to the client" allows for faster retrieval than from the "boot server," which may be located some distance from the client.

506. In a Birse system, combined with the caching techniques taught by Rothman, it would have been obvious to a POSITA to use a cache memory to cache blocks of data from the system volumes at the NC server upon a read request from one of the NC clients. Subsequently, a request for the same data from an NC client would cause the block device driver to retrieve the requested blocks of data from the cache memory rather than directly from the system volume.

507. It would have been obvious to a POSITA that the additional limitations of claim [8] describe nothing more than the standard operation of a cache memory. Performance of network booting is improved in Rothman is achieved by a cache

proximate to the NC client, which results in faster subsequent reads. Likewise, the Birse/Rothman system would have similar increased performance because read requests for the data blocks that are in the cache memory are provided from the cache, which is faster than the underlying data store. *See* §VII.D above.

508. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

509. In sum, the combination of Birse and Rothman discloses receiving a read request from at least one of said compute nodes [Birse's NC clients 150], wherein a first portion of the data requested is currently stored in said cache memory [Rothman's caching device]; and providing said first portion of said data to said at least one of said compute nodes from said cache memory."

10. Claim 9: The method as recited in claim 8 further comprising: updating said cache memory based on said read request.

510. *See* claim [8] in this Ground 3, in §XII.C.9 above.

511. As explained with respect to claim limitation [1.c] in §XII.C.2.e above, Rothman discloses caching techniques for a "boot image": "the boot image file(s) is/are cached at the network device or gateway server proximate to the client," and are obtained "from the caching device … rather than the boot server" on subsequent requests. EX1007 (Rothman) at 4:19-22, 8:56-58. In a Birse system, combined with the caching techniques taught by Rothman, it would have been obvious to a POSITA

to use a cache memory to cache blocks of data from the system volumes at the NC server upon a read request from one of the NC clients.

512. It would have been obvious to a POSITA that the additional limitations of claim [9] describe nothing more than the standard operation of a cache memory. In the event of a cache miss, the data requested is retrieved from the backing store and stored in the cache for subsequent faster reading from the cache. Upon a subsequent request for the same data (and provided that the data has not been evicted from the cache), the data is retrieved from the faster cache rather than the backing store. *See* §VII.D above.

513. A POSITA would have been motivated to combine Birse and Rothman for the reasons discussed in §XII.C.1 above.

514. In sum, the combination of Birse and Rothman discloses "updating said cache memory [Rothman's caching device] based on said read request."

11. Claim 10: The method as recited in claim 9 wherein a second portion of the data requested is not currently stored in said cache memory and said updating comprises: caching said second portion in said cache memory; and removing the least recently accessed data from said cache memory if the amount of data in said cache memory is above a threshold value.

515. *See* claims [2] (§XII.C.3 above) and [9] (§XII.C.10 above) in this Ground 3.

12. Claim 11: The method as recited in claim 7 further comprising: merging the blocks of said root image with the blocks of respective leaf images to create cohesive respective application environments.

516. *See* claim [4] (§XII.C.5 above) and claim limitations [7.a]-[7.c] (§§XII.C.8.b-e above) in this Ground 3.

13. Claim 12: The method as recited in claim 11 wherein said merging occurs at an operational level between file systems of the respective compute nodes and said first storage unit, respective second storage units, and said cache memory.

517. *See* claims [5] (§XII.C.6 above) and [11] (§XII.C.12 above) in this Ground 3.

14. Claim 13: The method as recited in claim 7 wherein said first storage unit is read-only.

518. *See* claim [6] (§XII.C.7 above) and claim limitations [7.a]-[7.c] (§§XII.C.8.b-e above) in this Ground 3.

D. Ground 4: Birse in View of Murphy Renders Claims 14-27 Obvious

1. Motivation to Combine Birse with Murphy

519. A POSITA would have been motivated to combine Birse and Murphy for at least the following five reasons.

520. **First**, the combination of Birse and Murphy represents the use of known techniques according to known methods in the same way to yield predictable results. Birse and Murphy both describe systems wherein multiple systems share a base file system. For example, in Birse, the "NC clients 150" share "system

volumes." Likewise, in Murphy, the "client computers 202" may share a single "boot image 500."

521. The Birse and Murphy combination would merely involve using Murphy's indexing in Birse's system, both of which were known, according to Murphy's methods for creation of an index for a boot image. For example, Birse teaches the use of a "system volumes" to store data that is common to multiple NC clients. *See* EX1006 (Birse) at 14:27-34, 14:41-44 ("[T]he NC server 170 contains a read-only core system volume image 922 and a read-write user system volume image 924 … The core system volume 922 preferably contains those parts of the system that do not need to be written back to during system operation. One goal of the core system volume 922 is to provide all the system components that are mandatory for system operation…. The user system volume 924 contains all the user-configurable system components, including preferences and all the system additions installed by application software, such as application-installed extensions and libraries.").

522. Murphy describes the creation of an index file from a boot image that may be shared by multiple client computers. *See* EX1005 (Murphy) at 14:60-62 ("[T]he index file may be created from the image file, which may be stored on server 206 or on client 202."); *id.*, 14:67-15:5 ("[A] single image file may be created for one or more client computers 202 [that] may be downloaded from server 20[6]

during the pre-boot process, or as necessary to aid in system configuration and maintenance.").

523. A POSITA would have recognized the similarity between Birse's "system volumes" and Murphy's "single image" file for use with "multiple client computers," and therefore would have found it obvious to create an index (as taught by Murphy) for the "system volumes" in Birse's system, to be shared between the NC clients that utilize the shared system volumes, because such sharing would be more efficient than creating separate indexes for each of Birse's NC clients.

524. **Second**, a POSITA would have been motivated to combine Birse with Murphy, because Murphy identifies a known problem (*e.g.*, file management) and provides a solution to address the problem (*e.g.*, creation of an index that contains a listing of the relevant files, the size of the files, the file location and/or other information about the files).

525. Birse describes techniques for a file system that uses a set of shared system volumes and one or more shadow volumes for each NC client that store modifications that are made to the system volumes. A POSITA would have been motivated to incorporate standard, well-known file-management techniques (such as the indexing that is described by Murphy) in the design of a file system such as the one described by Birse.

526. **Third**, a POSITA would have been motivated to use Murphy's indexing, because it would improve Birse's system/shadow volume storage system. Using Murphy's indexing to improve Birse's system would enable improved file management, including searching and file location. That is, the combination would merely be a use of a known technique (*e.g.*, indexing) to improve similar methods (*e.g.*, Birse's system/shadow volumes) in the same way (*e.g.*, creating an index containing information about the files stored in the system).

527. **Fourth**, a POSITA would have been motivated to combine Birse with Murphy, because the combination would involve applying a known technique (*e.g.*, using Murphy's indexing) to a known device (*e.g.*, Birse's system/shadow volumes), which was ready for improvement to yield predictable results. For example, it would be a trivial change to add indexing to Birse's system volumes, which are located at an "NC server 170" that is accessible by each of the NC clients. *See* EX1006 (Birse) at 5:10-11 ("The NC server contains one or more system volumes.").

528. **Fifth**, indexing and its benefits were well-known by 2004. *See* §VII.F above. Adding indexing (as taught by Murphy) to the Birse system would have provided the known benefits of more efficient file searching/locating. Accordingly, modifying the Birse system to use indexing as taught by Murphy is simply the use of a known technique (indexing) to improve similar systems (Birse's network

computing system using system/shadow volumes) to achieve the known benefits of indexing.

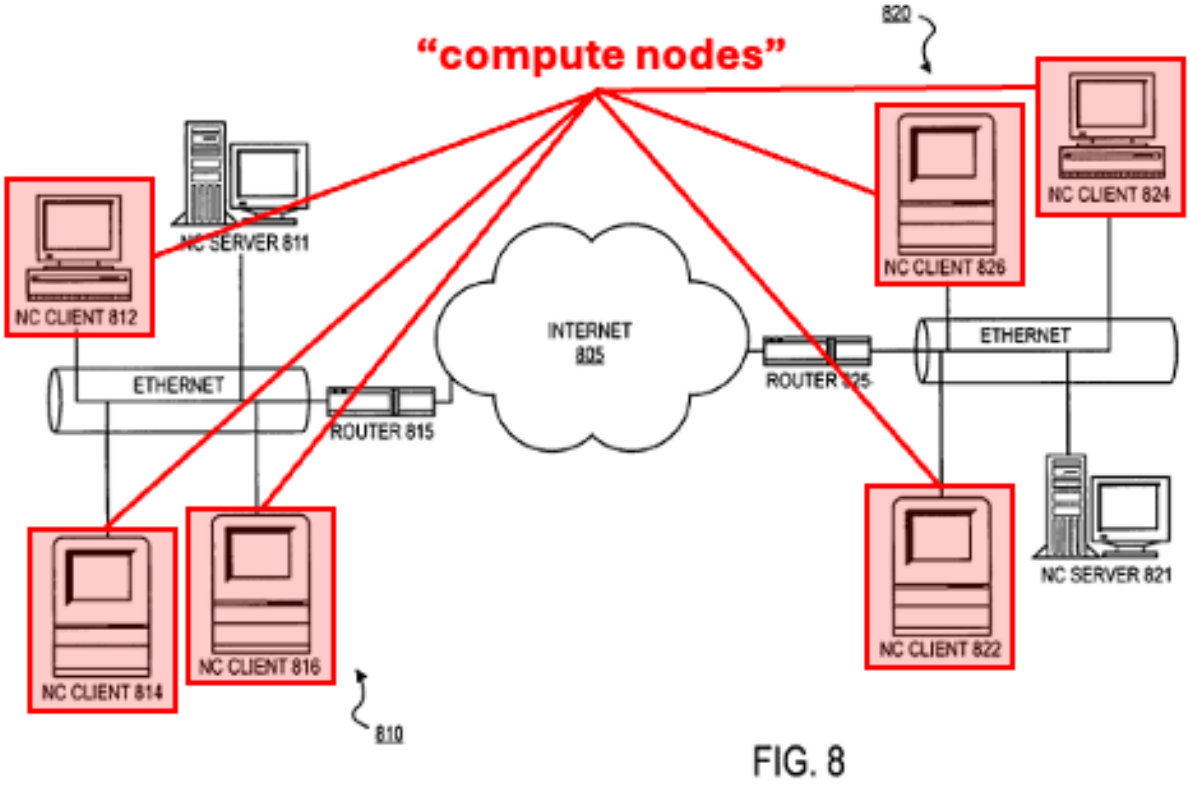
529. Finally, a POSITA would also have had a reasonable expectation of success in combining Birse with Murphy. Combining Birse with Murphy would merely involve performing indexing in the Birse system. Murphy describes the creation of an "index file" for "a single image file [that] may be created for one or more client computers," which is analogous to Birse's system volumes. *See id.*, 5:10-13 ("The system volumes 174 include a protected, read-only, master copy of the operating system software.").

2. Claim 14

a. [14.pre] A system for indexing file systems for a plurality of compute nodes, the system comprising:

530. As an initial matter, a POSITA would have known that indexing is an old and common technique used well before 2004. *See* §VII.F above.

531. As explained in §XII.C.2.a above with respect to claim limitation [1.pre] in Ground 3, Birse relates to an "NC system" that includes "one or more NC clients," (*i.e.*, "a plurality of compute nodes"), and "one or more NC servers" that "may provide access to a copy of the operating system to an NC client in response to a boot request." EX1006 (Birse) at 4:31-44, Figure 8 (reproduced and annotated below).



Birse, Figure 8 (annotated)

532. Murphy discloses indexing: "In various embodiments, a file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may to [*sic*] contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such embodiments, the index file may be created from the image file, which may be stored on server 206 or on client 202," and "a

single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:50-62, 14:67-15:2.

533. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

534. In sum, the combination of Birse and Murphy discloses "a system for indexing file systems [Murphy's index for an image file] for a plurality of compute nodes [Birse's NC clients 150]."

b. [14.a] a first storage unit configured to store blocks of a root image of said compute nodes;

535. *See* claim limitation [1.a] in Ground 3, in §XII.C.2.b above.

c. [14.b.i] a plurality of second storage units configured to store leaf images of respective compute nodes,

536. *See* claim limitation [1.b.i] in Ground 3, in §XII.C.2.c above.

d. [14.b.ii] said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of said root image, wherein said leaf images of respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes; and

537. *See* claim limitation [1.b.ii] in Ground 3, in §XII.C.2.d above.

e. [14.c] a plurality of union block devices corresponding to said compute nodes, said union block devices configured to interface between said compute nodes and said first and second storage units to distribute said file systems to said compute nodes, wherein said union block devices are configured to create said file systems by merging the blocks of said root image stored on the first storage unit with the blocks of respective leaf images stored on respective second storage units,

538. *See* claim [4] in Ground 3, in §XII.C.5 above.

f. [14.d] and wherein further at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

539. The combination of Birse and Murphy discloses that at least one of said compute nodes is configured to index said root image and provide the indexing results to another of said compute nodes.

540. Specifically, Murphy discloses the indexing of an "image file" (a root image) and the ability to provide the indexing results to another client computer by storing an "index file" on a server: "[A] file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may to [*sic*] contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such

embodiments, the index file may be created from the image file, which may be stored on server 206 or on client 202," and "a single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:52-62, 14:67-15:2.

541. Where a single "image file" is used for multiple client computers, it "may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance." *Id.*, 15:2-5.

542. It would have been obvious to a POSITA to incorporate the indexing taught by Murphy with the Birse system. As explained above, Birse teaches the use of "one or more system volumes" that "include a protected, read-only, master copy of the operating system software" (*i.e.*, "a root image"). EX1006 (Birse) at 5:9-14. A POSITA would have recognized the similarity between Birse's "system volumes" and Murphy's "single image" file for use with "multiple client computers." Accordingly, a POSITA would have found it obvious to create an index for Birse's "system volumes" to be shared between all of the NC clients that access the "system volumes."

543. Birse's "system volumes" are located within an "NC server" that is accessible by all of the NC clients that utilize the shared system volumes, similar to how Murphy's "server 206" stores the shared image file for access by multiple client computers. Thus, it would have been obvious for a POSITA to store an index for

the "system volumes" on the "NC server in order for the index to be provided to or accessed by the NC clients.

544. Moreover, a POSITA would have been aware of well-known techniques such as indexing and the sharing of data over networks. *See* §§VII.C and VII.F above.

545. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

546. In sum, the combination of Birse and Murphy discloses that "at least one of said compute nodes [Birse's NC clients 150] is configured to index said root image [Birse's system volumes 922, 924] and provide the indexing results [Murphy's index file] to another of said compute nodes [via storage of the index file at Birse's NC server 170 for access by all of the NC clients 150]."

3. Claim 15: The system as recited in claim 14 wherein said first storage unit and said second storage units are contained within a single storage appliance.

547. *See* claim limitations [14.pre]-[14.d] (§§XII.D.2.a-f above-) in this Ground 4, and claim [3] (§XII.C.4 above) in Ground 3.

4. Claim 16: The system as recited in claim 14 further comprising: a plurality of union block devices configured to interface between respective compute nodes and said first storage unit and respective second storage units, said union block devices con figured to distribute application environments to the compute nodes, wherein said union block devices are configured to create said application environments by merging the blocks of said root image with the blocks of respective leaf images.

548. *See* claim limitations [14.pre]-[14.d] (§§XII.D.2.a-f above) in this Ground 4, and claim [4] (§XII.C.5 above) in Ground 3.

5. Claim 17: The system as recited in claim 16 wherein said union block devices comprise low-level drivers for interfacing between the file systems of respective compute nodes and said first storage unit, respective second storage units, and said cache.

549. *See* claim [16] (§XII.D.4 above) in this Ground 4, and claim [5] (§XII.C.6 above) in Ground 3.

6. Claim 18: The system as recited in claim 14 wherein said first storage unit is read-only.

550. *See* claim limitations [14.pre]-[14.d] (§§XII.D.2.a-f above) in this Ground 4, and claim [13] (§XII.C.14 above) in Ground 3.

7. Claim 19

a. [19.pre] A method for indexing file systems for a plurality of compute nodes, comprising:

551. *See* claim limitation [14.pre] (§XII.D.2.a above) in this Ground 4.

b. [19.a] storing blocks of a root image of said compute nodes on a first storage unit;

552. *See* claim limitation [14.a] (§XII.D.2.b above) in this Ground 4.

c. [19.b.i] storing leaf images for respective compute nodes on respective second storage units,

553. *See* claim limitation [14.b.i] (§XII.D.2.c above) in this Ground 4.

d. [19.b.ii] said leaf images comprising only additional data blocks not previously contained in said root image and changes made by respective compute nodes to the blocks of the root image, wherein said leaf images for respective compute nodes do not include blocks of said root image that are unchanged by respective compute nodes;

554. *See* claim limitation [14.b.ii] (§XII.D.2.d above) in this Ground 4.

e. [19.c] merging the blocks of said root image with the blocks of respective leaf images stored on respective second storage units to create respective file systems for respective compute nodes;

555. *See* claim limitation [14.c] (§XII.D.2.e above) in this Ground 4.

f. [19.d] receiving indexing results pertaining to said root image from one of said compute nodes; and providing said indexing results to the others of said compute nodes.

556. *See* claim limitation [14.d] (§XII.D.2.f above) in this Ground 4.

8. Claim 20: The method as recited in claim 19 further comprising: storing said indexing results on a shared storage unit.

557. *See* claim limitations [19.pre]-[19.d] (§§XII.D.7.a-f above) in this Ground 4.

558. As explained in §XII.D.2.f above with respect to claim limitation [14.d] in this Ground 4, Murphy discloses the indexing of an "image file" (a root image) and the ability to provide the indexing results to another client computer by storing an "index file" on a server: "[A] file level image of the storage device is created by, for example, scanning the hard drive (or a particular portion of the hard drive, such as that portion storing operating system files or other files of interest) to create an index of the files. The index may to [*sic*] contain a listing of the relevant files, the size of the files, the file location and/or other information about the files. A separate disk image that includes the files themselves may also be created. In such embodiments, the index file may be created from the image file, which may be stored on server 206 or on client 202," and "a single image file may be created for one or more client computers 202." EX1005 (Murphy) at 14:52-62, 14:67-15:2.

559. Where a single "image file" is used for multiple client computers, it "may be downloaded from server 20[6] during the pre-boot process, or as necessary to aid in system configuration and maintenance." *Id.*, 15:2-5.

560. Thus, it would have been obvious to a POSITA to store an "index file" of Birse's system volumes within Birse's NC server, which, in one embodiment, stores both the system volumes and the shadow volumes.

561. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

562. In sum, the combination of Birse and Murphy discloses "storing said indexing results [Murphy's index file] on a shared storage unit [Birse's NC server 170]."

9. Claim 21: The method as recited in claim 19 wherein said merging occurs at an operational level between respective file systems of the compute nodes and said first storage unit and respective second storage units.

563. *See* claim limitations [19.pre]-[19.d] (§§XII.D.7.a-f above) in this Ground 4, and claim [5] (§XII.C.6 above) in Ground 3.

10. Claim 22: The method as recited in claim 19 wherein said first storage unit is read-only.

564. *See* claim limitations [19.pre]-[19.d] (§§XII.D.7.a-f above) in this Ground 4, and claim [6] (§XII.C.7 above) in Ground 3.

11. Claim 23

a. [23.pre] A computer-readable storage medium having instructions stored thereon that, in response to execution by at least one computing device, cause the at least one computing device to:

565. *See* claim limitation [1.pre] (§XII.C.2.a above) in Ground 3.

b. [23.a] receive data blocks of a file system, said data blocks comprising a root image portion and leaf image portion,

566. *See* claim limitations [1.a] (§XII.C.2.b above) and [1.b.i] (§XII.C.2.c above) in Ground 3.

c. [23.b] said leaf image portion comprising only additional data blocks not previously contained in said root image portion and, changes made by said first compute node to the blocks of said root image, wherein said leaf image portion does not include blocks of said root image that are unchanged by said first compute node,

567. *See* claim limitation [1.b.ii] (§XII.C.2.d above) in Ground 3.

d. [23.c] wherein said file system is the result of merging said root image portion and said leaf image portion together at the block-level;

568. *See* claim [4] (§XII.C.5 above) in Ground 3.

e. [23.d] index said root image portion; and provide the results of said indexing to a second compute node,

569. *See* claim limitation [14.d] (§XII.D.2.f above) in this Ground 4.

f. [23.e] wherein said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node.

570. As an initial matter, I note that there is no antecedent basis for "said logic encoded in the one or more tangible media" in claim 23. I have been instructed by Petitioner to interpret this phrase as referring to the "instructions" recited in [23.pre] in §XII.D.11.a above.

571. Birse discloses that said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node (NC client 150): "The NC client also includes a file system 152, a block device driver 154, a network stack 156, and a network device driver 158, each of which

may include hard-wired circuitry or **machine-executable instructions** or a combination thereof. Furthermore, at least a portion of such hard-wired circuitry and/or **machine-executable instructions** may be shared between a combination of the file system 152, the block device driver 154, the network stack 156, and the network device driver 158." EX1006 (Birse) at 7:6-20; *see also id.*, 3:22-27 ("The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the steps.").

572. It would have been obvious to a POSITA that this limitation describes the general operation of an "NC client"—a type of computer—that executes instructions to function. *See* §VII.A above.

573. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

574. In sum, the combination of Birse and Murphy discloses that "said logic encoded in the one or more tangible media comprise computer executable instructions executed by the first compute node [Birse's NC client 150]."

12. Claim 24: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to store said results of said indexing on a shared storage unit accessible by said second compute node.

575. *See* claim limitations [23.pre]-[23.e] (§§XII.D.11.a-f above) and claim [20] (§XII.D.8 above) in this Ground 4.

13. Claim 25: The computer-readable storage medium of claim 23, wherein the instructions further cause the at least one computing device to index said leaf image portion.

576. *See* claim limitations [23.pre]-[23.e] (§§XII.D.11.a-f above-XXX) in this Ground 4.

577. As explained with respect to claim limitation [14.d] in §XII.D.2.f above, it would have been obvious to a POSITA to incorporate the indexing taught by Murphy with the Birse system, and to create an index for Birse's "system volumes" to be shared between all of the NC clients 150 that utilize the system volumes.

578. It also would have been obvious to a POSITA to incorporate indexing of each of the "shadow volumes" corresponding to each of Birse's NC clients, such that any changes (or additions) to the user system volumes 924 would also be indexed, since it is the combination of the system volumes and the shadow volumes that form the complete file system for each NC client.

579. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

580. In sum, the combination of Birse and Murphy discloses that "the instructions further cause the at least one computing device [Birse's NC client 150] to index [Murphy's index file] said leaf image portion [modifications that are made to the user system volume 924]."

14. Claim 26: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

581. *See* claim [25] in this Ground 4, in §XII.D.13 above.

582. It would have been obvious to a POSITA to re-index the shadow volumes periodically as they are modified by their respective NC clients 150 to ensure that the index is up-to-date and includes all modifications and newly created files. Furthermore, it would have been obvious to a POSITA that, because the complete file system for each NC client is the combination of the operating system software stored in the system volumes along with the modifications to the original operating system that are stored in the shadow volumes, that the indexes of the system volumes and shadow volumes would need to be combined (*i.e.*, "merged" as required by claim [26]) in order to have a complete index for the full file system. *See also* §VII.F above.

583. A POSITA would have been motivated to combine Birse and Murphy for the reasons discussed in §XII.D.1 above.

584. In sum, the combination of Birse and Murphy discloses that "the instructions further cause the at least one computing device [Birse's NC client 150] to re-index [Murphy's index file] said file system by re-indexing said leaf image portion [modifications to Birse's system volumes] and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion [operating system software stored in Birse's system volumes]."

15. Claim 27: The computer-readable storage medium of claim 25, wherein the instructions further cause the at least one computing device to re-index said file system by re-indexing said leaf image portion and merging the results of said re-indexing of said leaf image portion with said results of said indexing of said root image portion.

585. *See* claim limitations [23.pre]-[23.e] (§§XII.D.11.a-f above) in this Ground 4, and claim [6] (§XII.C.7 above) in Ground 3.

XIII. SECONDARY CONSIDERATIONS OF NON-OBVIOUSNESS

586. I have seen no evidence in either the '844 patent's prosecution history or elsewhere supporting any secondary considerations arguments, or evidence of nexus of such alleged evidence to the challenged claims. To the extent patent owner or its expert asserts the existence of any secondary considerations in its responses, I reserve the right to address any such evidence.

XIV. CONCLUSION

587. The findings and opinions set forth in this declaration are based on my work and examinations to date.

588. In signing this declaration, I recognize that the declaration will be filed as evidence in an IPR before the PTAB. I also recognize that I may be subject to cross-examination in this case, and that cross-examination will take place within the United States. If cross-examination is required of me, I will appear for cross-examination within the United States during the time allotted for cross-examination.

I hereby declare under penalty of perjury that the foregoing is true and correct, and that all statements made herein of my knowledge are true, and that all statements made on information and belief are believed to be true, and that these statements were made with the knowledge that willful false statements so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code.

Dated: April 28, 2025



Erez Zadok, Ph.D.

EREZ ZADOK, CURRICULUM VITAE

Erez Zadok
Computer Science Department
349 New Computer Science
Stony Brook University
Stony Brook, NY 11794-2424

Phone: +1 631 632 8461 (office)
Fax: +1 631 632 8243
Timezone: US/Eastern
Email: ezk@cs.stonybrook.edu
Web: https://www.cs.sunysb.edu/~ezk

RESEARCH INTERESTS

Operating systems with a special focus on file systems, storage, clouds, big data, hardware/architecture, encryption, security, benchmarking, performance analysis and optimization, energy efficiency, and system administration.

EDUCATION

May 2001 Ph.D., Computer Science, Columbia University, New York, NY,
FiST: A System for Stackable File-System Code Generation.

Sep 1997 M.Phil., Computer Science, Columbia University, New York, NY

Oct 1994 M.S., Computer Science, Columbia University, New York, NY
Discovery and Hot Replacement of Replicated Read-Only File Systems, with Application to Mobile Computing

May 1991 B.S., Computer Science, Columbia University, New York, NY

May 1982 Certified Technician, Electrical Engineering, Holtz College, Israel

PROFESSIONAL EXPERIENCE

Aug 2024–present	Graduate Program Director, Computer Science Department, Stony Brook University
Jul 2024–present	Senator, University Faculty Senate, representative of the Computer Science Department, Stony Brook University
Aug 2017–Jul 2024	Graduate Academic Adviser, Computer Science Department, Stony Brook University
Jan 2016–present	Professor, Computer Science Department, Stony Brook University
Jan 2007–2015	Associate Professor, Computer Science Department, Stony Brook University
Jan 2001–Jan 2007	Assistant Professor, Computer Science Department, Stony Brook University
2013–present	Director, Smart Energy Technologies (SET) Faculty Cluster, Stony Brook University
2025–present	President; Zadok Consulting LI Inc., DBA "Zadoks Consulting Services"
2013–present	Managing Member; Zadoks Consulting, LLC.
2009–2010	Consultant; CTERA Networks, Inc.
2009–2019	Consultant; Packet General Networks, Inc.
1991–2000	Graduate Research Assistant, Computer Science Department, Columbia University
1999–2000	Director of Software Development, HydraWEB Technologies, Inc.
1994–1998	Project Leader, HydraWEB Technologies, Inc.

Docker EX1003
Page 239 of 310

1990–1998	Consultant, SOS Corporation
1997	Manager of Computing Facilities, Computer Science Department, Columbia University
1991–1998	Technical Staff Member, Computer Science Department, Columbia University
1989–1991	Assistant Lab Manager, Academic Information Systems, Columbia University
1987–1989	Student Consultant, Academic Information Systems, Columbia University
1984–1986	National Army Service, Israeli Air Force, Israel
1982–1984	Programmer, Commodore Israel, Tel-Aviv, Israel
1981–1984	Computer Lab Manager, Holtz College, Tel-Aviv, Israel

PERSONAL

Born December 4, 1964, Tel-Aviv, Israel.

Married, one child.

Citizenships: U.S.A and Israel

Fluent in English and Hebrew

Member: ACM, ACM SIGOPS, IEEE, IEEE Computer Society, USENIX

Affiliate: Storage Systems Research Center (SSRC), Jack Baskin School of Engineering, University of California, Santa Cruz, California.

Member: The I/O Traces, Tools and Analysis (IOTTA) Technical Work Group (TWG), part of the the Storage Networking Industry Association (SNIA).

FUNDING

Oct 2022–present	*CNS Core: Large: Systems and Verifiable Metrics for Sustainable Data Centers*. NSF. $1,500,000 (SBU share $928,403), 4 years. Co-PI with four other SBU faculty, collaborative with Binghamton University and Penn State University.
Oct 2022–present	*Collaborative Research: CyberTraining: Implementation: Medium: FOUNT: Scaffolded, Hands-On Learning for a Data-Centric Future*. NSF. $996,548 (SBU share $174,905), 3 years. Sole PI at SBU. Collaborative with U. Chicago (lead), NYU, Northern Illinois U., and UC San Diego (6 collaborators total).
2022	*CNS Core: Medium: Optimizing Storage Caches via Adaptive and Reconfigurable Tiering*. NSF Research Experiences for Undergraduates (REU) supplement. $16,000. Lead-PI with one other SBU faculty, collaborative with Emory University.
2022	*SCC-IRG Track 1: Smart Aging: Connecting Communities Using Low-Cost and Secure Sensing Technologies*. NSF Research Experiences for Undergraduates (REU) supplement. $24,000. Co-PI with four other SBU faculty.
2022	*CNS Core: Medium: Optimizing Storage Caches via Adaptive and Reconfigurable Tiering*. NSF Research Experiences for Undergraduates (REU) supplement. $16,000. Lead-PI with one other SBU faculty, collaborative with Emory University.

Docker EX1003
Page 240 of 310

2022	*CNS Core: III: Medium: Collaborative Research: Optimizing and Understanding Large Parameter Spaces in Storage Systems*. NSF Research Experiences for Undergraduates (REU) supplement. $16,000. Lead PI with one other.
Oct 2021–present	*CNS Core: Medium: Optimizing Storage Caches via Adaptive and Reconfigurable Tiering*. NSF. $800,000 (SBU share $533,333), 3 years. Lead-PI with one other SBU faculty, collaborative with Emory University.
Oct 2021–present	*CNS Core: Medium: Secure, Reliable, and Efficient Long-Term Storage*. NSF. $1,198,842 (SBU share $717,303), 4 years. Lead-PI with one other SBU faculty, collaborative with UC Santa Cruz.
2021–2022	*Kubernetes/Containerizations on GPU Clusters for AI*. SUNY-IBM AI Collaborative Research Alliance. $100,000. Co-PI with two others.
2020–2021	*Automated Cross-Validation of TLS 1.3 Implementations*. Facebook Faculty Research Award. $50,000. Lead PI with two others.
Oct 2020–present	*SCC-IRG Track 1: Smart Aging: Connecting Communities Using Low-Cost and Secure Sensing Technologies*. NSF Smart and Connected Communities. $1,700,126, 4 years. Co-PI with four other SBU faculty.
Sep 2020–present	*Recruitment and Retention of Women Undergraduates*. Center for Inclusive Computing Diversity Initiative, by Northeastern University and Pivotal Ventures. $295,742, 4 years. Co-PI with four other SBU faculty.
2020	*FMitF: Track I: NLP-Assisted Formal Verification of the NFS Distributed File System Protocol*. NSF Research Experiences for Undergraduates (REU) supplement. $16,000. Lead PI with two others.
2020	*CNS Core: III: Medium: Collaborative Research: Optimizing and Understanding Large Parameter Spaces in Storage Systems*. NSF Research Experiences for Undergraduates (REU) supplement. $8,000. Lead PI with one other.
Oct 2019–present	*CNS Core: III: Medium: Collaborative Research: Optimizing and Understanding Large Parameter Spaces in Storage Systems* NSF. $1,088,017 (SBU share $823,142), 4 years. Lead-PI with one other SBU faculty, collaborative with Harvey Mudd College.
Oct 2019–present	*FMitF: Track I: NLP-Assisted Formal Verification of the NFS Distributed File System Protocol* NSF. $748,300, 3 years. Lead-PI with two other SBU faculty.
2019	*CI-SUSTAIN: National File System Trace Repository*. NSF Research Experiences for Undergraduates (REU) supplement. $8,000. Sole PI.
2019	*Study of a Novel Non-Wearable Respiration and Heart Rate Sensor in Cardiopulmonary Exercise Testing*. Stony Brook College of Engineering and Applied Sciences, SEED grant. $15,000, Co-PI with five others.
2019	*Realizing the Full Performance and Parallelization Potential of Modern Storage Architectures for Big Data Applications*. Stony Brook Research Foundation, SEED grant. $60,000, Co-PI with Anshul Gandhi.
2018	*Storage/Deduplication research*. Dell-EMC Corporation. $25,000, Single PI.

Docker EX1003
Page 241 of 310

Jun 2017–present	*CI-SUSTAIN: National File System Trace Repository*. NSF. $129,867 (SBU share), 3 years. Co-PI with lead institution Harvey Mudd College.
2017–2022	*I/UCRC Phase II: Center for Visual and Decision Informatics (CVDI) Site at SUNY Stony Brook*. NSF. $400,000, 4 years. Co-PI with A. Kaufman, K. Mueller, H. Schwartz, and D. Samaras.
2017	Dell-EMC Corporation. *Storage/Deduplication research*, $25,000, Single PI.
2016–2021	*NRT-DESE: Interdisciplinary Graduate Training to Understand and Inform Decision Processes Using Advanced Spatial Data Analysis and Visualization (STRIDE)*. NSF. $2,993,930, 5 years. Senior personnel.
2016–2019	*Early Detection of User-impersonating Attackers using Multilayer Tripwires*, U.S. Office of Naval Research (ONR). $586,215, 3 years. Co-PI with Nick Nikiforakis.
Jun 2016–2019	*EAGER: Elastic Multi-layer Memcached Tiers* NSF. $257,165, 2 years. Co-PI with Anshul Gandhi.
2016	EMC Corporation. *Storage/Deduplication research*, $25,000, Single PI.
2015	EMC Corporation. *Storage/Deduplication research*, $25,000, Single PI.
Feb 2015	*Student Travel Support for the 13th USENIX Conference on File and Storage Technologies (FAST 2015)*. NSF. $20,000, 1 year. Sole PI.
Sep 2014–2017	*Adaptive Runtime Verification and Recovery for Mission-Critical Software*. U.S. Air Force Office of Scientific Research (AFOSR). $620,861, 3 years. Co-PI with Scott A. Smolka and Scott D. Stoller. (Collaboration with NASA JPL.)
Jan 2014–2016	*Smarter Electric Grid Research, Innovation, Development, Demonstration, Deployment Center (SGRID3)*. Brookhaven Science Associates LLC (BNL), $236,397, 1 year. Lead-PI with one other Stony Brook Co-PI.
2014	EMC Corporation. *Storage/Deduplication research*, $25,000, Sole PI.
Jun 2013–2017	*CSR: Medium: Collaborative Research: Workload-Aware Storage Architectures for Optimal Performance and Energy Efficiency*. NSF. $513,900 (SBU share, total budget $1,000,000), 3 years. Lead-PI with one other Stony Brook Co-PI, and two more institutions (Harvard U. and Harvey Mudd College).
Jan 2013–2017	*BIGDATA: Small: DCM: Collaborative Research: An efficient, versatile, scalable, and portable storage system for scientific data containers*. NSF. $444,267 (SBU share, total budget $746,290), 3 years. Lead-PI with two other Stony Brook Co-PIs, and two more institutions (Brandeis U. and Louisiana State U.).
Sep 2013–2017	*CRI-CI-ADDO-EN: National File System Trace Repository*. NSF. $37,018 (SBU share, total budget $167,817), 3 years. Co-PI with lead institution Harvey Mudd College.
2013	Western Digital Research award. *Shingled Magnetic Recording Disks Benchmarking*, $50,000, Single PI.
Sep 2012 – 2016	*NFS4Sec: An Extensible Security Layer for Network Storage*. NSF. $486,783, 3 years. Lead-PI with one other Co-PI.

Docker EX1003
Page 242 of 310

2012-2013	*Server-Class Performance vs. Energy Optimizations*. Government of Israel (GoI), Mission to the USA. $47,152, 1 year. Lead PI with one other Co-PI.
2011	NetApp Research award. *Dedup Workload Modeling, Synthetic Datasets, and Scalable Benchmarking*, $40,000, Single PI.
2010	NetApp Research award. *A Study of Network Storage Benefits using FLASH Hardware with Indexing Workloads*, $40,000, Single PI.
Nov 2010 – 2016	*Long Island Smart Energy Corridor*. Department of Energy (DOE), LIPA, and New York State. Collaboration between Stony Brook University, SUNY Farmingdale, and LIPA. $2,822,638, Co-PI.
Sep 2009 – Aug 2013	*Collaborative Research: Performance- and Energy-Aware HEC Storage Stacks*. NSF. $652,000, 3 years. Co-PI with Geoff Kuenning (Harvey Mudd College)
Sep 2009 – Aug 2013	*Collaborative Research: Secure Provenance in High End Computing Systems*. NSF. $564,972, 3 years. Co-PI with Radu Sion. Collaborative project with Patrick McDaniel (Penn State U.) and Marianne Winslett (UIUC).
Apr 2009 – Nov 2012	*Survivable Software*. U.S. Air Force Office of Scientific Research (AFOSR). $881,691, 39 months. Co-PI with Scott A. Smolka, Radu Grosu, Scott D. Stoller, and Klaus Havelund (NASA JPL).
Feb 2010	*Student Travel Support for the First USENIX Workshop on Sustainable Information Technology (SustainIT 2010)*. NSF. $10,000, 1 year. Lead PI.
2009	Network Appliance Research award. *Power use in Storage Servers.* $30,000 Single PI.
2008	*The Impact of Storage Software and Aging on Power Consumption*, **IBM Faculty award** (IBM T.J. Watson Labs). $20,000, one year. Single PI.
2008	Network Appliance Equipment gift. *A Study of User File Access Patterns.* $91,083 Single PI.
Sep 2007 – 2015	*Center for Information Protection: A Multi-University Industry/University Collaborative Research Center*. NSF. $250,147, 5 years. Co-PI with R. Sekar (PI), Tzi-Cker Chiueh, Scott Stoller, and Radu Sion.
Sep 2006 – Aug 2009	*CT-ISG: N3S: Networked Secure Searchable Storage with Privacy and Correctness Assurances*. NSF. $300,000, 3 years. Co-PI with Radu Sion.
Aug 2006 – Aug 2010	*File System Tracing, Replaying, Profiling, and Analysis on HEC Systems*. NSF. $760,252, 3 years. Lead PI with Klaus Mueller (Stony Brook) and Ethan Miller (UC Santa Cruz).
Jul 2006	*End-to-End File Server Security*, **IBM Faculty award** (IBM Haifa Research Labs). $20,000, one year. Single PI.
Jun 2006 – Aug 2010	*CSR—PDOS: Support for Atomic Sequences of File System Operations*. NSF. $561,727, 3 years. Lead PI with Margo Seltzer (Harvard University).
Jan 2006 – Dec 2006	*Secure File Systems*, NY State "Millennium" award, $204,528, one year. Co-PI with R. Sekar (PI), Tzi-Cker Chiueh, CR Ramakrishnan, Radu Sion, and Scott D. Stoller.

Docker EX1003
Page 243 of 310

Jul 2005 – Aug 2010	*CSR—AES: Runtime-Monitoring and Model Checking for High-Confidence Systems Software*. NSF. $830,000, 4 years. Lead PI with Radu Grosu, Y. Annie Liu, Scott Smolka, and Scott D. Stoller.
Sep 2005 – Aug 2004	*I/UCRC: A Plan for Developing a Multi-University Industry/University Collaborative Research Center on Cyber Security*. NSF. $9,987, one year. Co-PI with R. Sekar, Radu Sion Scott D. Stoller, and Tzi-Cker Chiueh.
Sep 2004 – Aug 2009	*Federal Cyber Service: Scholarship for Service (SFS)*. NSF. $2,459,061, 4 years. Co-PI with R. Sekar, Scott D. Stoller, I. V. Ramakrishnan, and Tzi-Cker Chiueh.
Sep 2003 – Aug 2008	*A Layered Approach to Securing Network File Systems*. NSF Trusted Computing Program (TC). $400,000, 3 years. Single PI.
Sep 2003 – Aug 2005	*Collaborative Research: Capacity Expansion in Information Assurance*. NSF Collaborative Research Proposal (CAP). $199,883, 2 years. Co-PI with R. Sekar, Scott D. Stoller, and I. V. Ramakrishnan.
Sep 2002 – Aug 2007	**CAREER**: *An In-Kernel Runtime Execution Environment for User-Level Programs.* NSF Next Generation Software Program (NGS). $400,000, 5 years. Single PI.
Jan 2003 – Dec 2003	*Assessing the Technological Basis for Enterprise Protection*, NIJ (CyberScience Lab) CSL. $90,563, one year. Co-PI with R. Sekar, Tzi-cker Chiueh, and Scott D. Stoller.
Sep 2002 – May 2003	New York State Strategic Partnership for Industrial Resurgence (SPIR), with Packet General Networks. *S3: Secure Shared Storage.* $94,581. Single PI.
May 2002 – Aug 2002	New York State Strategic Partnership for Industrial Resurgence (SPIR), with Packet General Networks. *A Secure and Scalable Network Appliance.* $55,676. Single PI.
Apr 2003	Microsoft Tablet PC Seed Award. $4,169. Single PI.
2002	HP/Intel IA-64/IPF Second Generation Equipment gift. *Linux Application Performance and File System Security.* $131,529. Single PI.
2001	HP/Intel IA-64/IPF Equipment gift. *Linux Network Scalability and File System Reliability.* $22,490. Single PI.
2001	Red Hat University software award.

AWARDS AND HONORS

2023	Best paper award. A. Merenstein, V. Tarasov, A. Anwar, S. Guthridge, and E. Zadok. F3: Serving Files Efficiently in Serverless Computing. In Proceedings of the 16th ACM International Systems and Storage Conference (SYSTOR '23).
2023	Elected as IEEE Senior Member
2021–2022	Chancellor's Award for Excellence in Scholarship and Creative Activities, State University of New York (SUNY).
2022	Provost's Excellent Mentor award, Stony Brook University.

Docker EX1003
Page 244 of 310

2021 Elected as ACM Distinguished Member for "Outstanding Scientific Contributions to Computing."

2021 Dean's "Millionaire" award (given lead PIs who secure more than one-million dollars of new funding in a given year), College of Engineering and Applied Sciences, Stony Brook University.

2020 Inducted as member of the Stony Brook University Chapter of the National Academy of Inventors (NAI).

2020 Best paper award. M. Wajahat, A. Yele, T. Estro, A. Gandhi, and E. Zadok. Distribution fitting and performance modeling for storage traces. In 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2019).

2019 Dean's "Millionaire" award (given lead PIs who secure more than one-million dollars of new funding in a given year), College of Engineering and Applied Sciences, Stony Brook University.

Sep 2011 Best Paper Award. S. D. Stoller and E. Bartocci and J. Seyster and R. Grosu and K. Havelund and S. A. Smolka and E. Zadok. Runtime verification with state estimation. In proceedings of the 2nd International Conference on Runtime Verification (RV'11).

Dec 2009 LISTnet's "Top 20 techies of Long Island" award.

Aug 2008 Service Award (for 2006–2008), Stony Brook University, Computer Science Department

2008 IBM Faculty award (IBM T.J. Watson Labs)

2007–2008 Chancellor's Award for Excellence in Teaching, State University of New York (SUNY).

2007–2008 President's Award for Excellence in Teaching, Stony Brook University.

Aug 2006 Research Excellence Award (for 2005–2006), Stony Brook University, Computer Science Department

Jul 2006 IBM Faculty award (IBM Haifa Research Labs)

Nov 2005 Best Short Paper Award. N. Joukov, A. Kashyap, G. Sivathanu, E. Zadok. Kefence: An Electric Fence for Kernel Buffers. In proceedings of the first ACM International Workshop on Storage Security and Survivability (StorageSS 2005), "The Paradigm Shift to Info-Centric Protection," held in conjunction with the 12th ACM Conference on Computer and Communications Security (CCS 2005).

May 2005 Best Paper Award. N. Joukov, A. Rai, and E. Zadok. Increasing Distributed Storage Survivability with a Stackable RAID-like File System. In proceedings of the 2005 IEEE/ACM Workshop on Cluster Security, in conjunction with the Fifth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid).

Aug 2004 Graduate Teaching and Research Award (for 2003–2004), Stony Brook University, Computer Science Department

May 2003 Promising Inventor Award, The State University of New York, Research Foundation

Sep 2002 NSF CAREER award, National Science Foundation

Jun 2001 Best Student Paper. M. G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya, and S. J. Stolfo. MEF: Malicious Email Filter — A UNIX Mail Filter that Detects Malicious Windows Executables. In *Proceedings of the Annual USENIX Technical Conference, FreeNIX Track*, pages 245–252, JUNE 2001.

Jun 2000 Usenix Student Stipend award

Jun 1999 Usenix Student Stipend award

1991 Dean's List

1991 NCR Best Undergraduate Research Award

1989–1990 Gussman Scholar

1988–1989 National Dean's List

1988–1989 Scheuer Foundation Presidential Scholar

1988 Dean's List

PUBLICATIONS

Books

E. Zadok. *Linux NFS and Automounter Administration*. Sybex, Inc., May 2001.

Journal and Magazine Articles

Tyler Estro, Mário Antunes, Pranav Bhandari, Anshul Gandhi, Geoff Kuenning, Yifei Liu, Carl Waldspurger, Avani Wildani, and Erez Zadok. Accelerating multi-tier storage cache simulations using knee detection. *Performance Evaluation*, 164:102410, May 2024.

S. Hussain, P. McDaniel, A. Gandhi, K. Ghose, K. Gopalan, D. Lee, Y. Liu, Z. Liu, S. Mu, and E. Zadok. Verifiable sustainability in data centers. *IEEE Security & Privacy*, (01):2–15, March 2024.

Anshul Gandhi, Dongyoon Lee, Zhenhua Liu, Shuai Mu, Erez Zadok, Kanad Ghose, Kartik Gopalan, Yu David Liu, Syed Rafiul Hussain, and Patrick Mcdaniel. Metrics for sustainability in data centers. *SIGENERGY Energy Inform. Rev.*, 3(3):40–46, oct 2023.

Ibrahim "Umit" Akgun, Ali Selman Aydin, Andrew Burford, Michael McNeill, Michael Arkhangelskiy, and Erez Zadok. Improving storage systems using machine learning. *ACM Transactions on Storage (TOS)*, 19(1):1–30, Jan 2023.

Erez Zadok and Ada Gavrilovska. Running virtual pc (cpc) meetings. *;login: The USENIX Magazine*, 45(2):54–56, July 2020.

Muhammad Wajahat, Aditya Yele, Tyler Estro, Anshul Gandhi, and Erez Zadok. Analyzing the distribution fit for storage workload and internet traffic traces. *Performance Evaluation*, pages 102–121, 2020.

George Amvrosiadis, Ali R. Butt, Vasily Tarasov, Erez Zadok, and Ming Zhao. Selected results of the workshop on data storage research 2025. *;login: The USENIX Magazine*, 44(4):24–28, December 2019.

Bharath Kumar Reddy Vangoor, Prafful Agarwal, Manu Mathew, Arun Ramachandran, Swaminathan Sivaraman, Vasily Tarasov, and Erez Zadok. Performance and resource utilization of FUSE user-space file systems. *ACM Transactions on Storage (TOS)*, 15(2), May 2019.

Zhen "Jason" Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. Cluster and single-node analysis of long-term deduplication patterns. *ACM Transactions on Storage (TOS)*, 14(2), May 2018.

Ming Chen, Geetika Bangera, Dean Hildebrand Farhaan Jalia, Geoff Kuenning, Henry Nelson, and Erez Zadok. vNFS: Maximizing NFS performance with compounds and vectorized I/O. *ACM Transactions on Storage (TOS)*, 13(7), September 2017.

Vasily Tarasov, Erez Zadok, and Spencer Shepler. Filebench: A flexible framework for file system benchmarking. *;login: The USENIX Magazine*, 41(1):6–12, March 2016.

Z. Li, M. Chen, A. Mukker, and E. Zadok. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive. *ACM Transactions on Storage (TOS)*, 11(3), July 2015.

Ming Chen, Dean Hildebrand, Geoff Kuenning, Soujanya Shankaranarayana, Bharat Singh, and Erez Zadok. Is NFSv4.1 ready for prime time? *;login: The USENIX Magazine*, 40(3):6–12, June 2015.

Zhiyuan Zhang, Kevin T. McDonnell, Erez Zadok, and Klaus Mueller. Visual correlation analysis of numerical and categorical data on the correlation map. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 21(2), February 2015.

Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. 5(11):1627–1637, August 2012.

J. Seyster, K. Dixit, X. Huang, R. Grosu, K. Havelund, S. A. Smolka, S. D. Stoller, and E. Zadok. InterAspect: Aspect-oriented instrumentation with GCC. *Formal Methods in System Design*, August 2012.

X. Huang, J. Seyster, S. Callanan, K. Dixit, R. Grosu, S. A. Smolka, S. D. Stoller, and E. Zadok. Software monitoring with controllable overhead. *International Journal on Software Tools for Technology Transfer (STTT)*, 14(3):327–347, 2012.

Priya Sehgal, Vasily Tarasov, and Erez Zadok. Optimizing energy and performance for server-class file system workloads. *ACM Transactions on Storage (TOS)*, 6(3), September 2010.

E. Zadok, V. Tarasov, and P. Sehgal. The case for specialized file systems, or, fighting file system obesity. *;login: The USENIX Magazine*, 35(1):38–40, February 2010.

M. T. Dougherty, M. J. Folk, E. Zadok, H. J. Bernstein, F. C. Bernstein, K. W. Eliceiri, W. Benger, and C. Best. Unifying biological image formats with hdf5. *Communications of the ACM (CACM)*, 52(10):42–47, October 2009.

A. Traeger and E. Zadok. Notes on a nine year study of file system and storage benchmarking. *Byte and Switch*, July 2009. www.byteandswitch.com/storage/storage-management/notes-on-a-nine-year-study-of-file-system-and-storage-benchmarking.php.

A. Traeger, E. Zadok, E. L. Miller, and D. D. E. Long. Findings from the first annual storage and file systems benchmarking workshop. *;login: The USENIX Magazine*, 33(5):113–117, October 2008.

Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. A nine year study of file system and storage benchmarking. *ACM Transactions on Storage (TOS)*, 4(2):25–80, May 2008.

C. P. Wright, R. Spillane, G. Sivathanu, and E. Zadok. Extending ACID semantics to the file system. *ACM Transactions on Storage (TOS)*, 3(2):1–42, June 2007.

Erez Zadok, Rakesh Iyer, Nikolai Joukov, Gopalan Sivathanu, and Charles P. Wright. On incremental file system development. *ACM Transactions on Storage (TOS)*, 2(2):161–196, 2006.

C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and unix semantics in namespace unification. *ACM Transactions on Storage (TOS)*, 2(1):1–32, February 2006.

C. P. Wright and E. Zadok. Unionfs: Bringing file systems together. *Linux Journal*, 2004(128):24–29, December 2004.

E. Zadok. Using the Amd automounter. *Linux Journal*, 10(114):52–55, October 2003.

E. Zadok. Writing stackable file systems. *Linux Journal*, 05(109):22–25, May 2003.

W. Lee, W. Fan, M. Miller, S. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. *Journal of Computer Security*, 10(1–2):5–22, January 2002.

Refereed Conference and Workshop Papers

Dongsheng Luo, Raju Rangaswami, Amir Rahmati, and Erez Zadok. GitTemporalAI: Leveraging temporal knowledge graphs and LLMs for multi-agent repository intelligence. 2025. To appear in The First MARW: Multi-Agent AI in the Real World Workshop at AAAI 2025.

Mohammad Saqib Hasan, Sayontan Ghosh, Dhruv Verma, Geoff Kuenning, Erez Zadok, Scott Smolka, and Niranjan Balasubramanian. Handling open-vocabulary constructs in formalizing specifications: Retrieval augmented parsing with expert knowledge. In *Proceedings of the First Conference on Language Modeling (COLM 2024)*, Philadelphia, PA, October 2024.

Christopher Smith, Maliha Tabassum, Soumya Chowdary Daruru, Gaurav Kulhare, Arvin Wang, Ethan Miller, and Erez Zadok. Secure archival is hard... really hard. In *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '24)*, Santa Clara, CA, July 2024. ACM.

Alex Merenstein, Xinran Wang, Vasily Tarasov, Prajjawal Agarwal, Scott Guthridge, Kapil Thakkar, Katherine Wu, Ali Anwar, and Erez Zadok. Balancing costs and durability for serverless data. In *Proceedings of the 38th International IEEE Symposium on Mass Storage Systems and Technologies (MSST '24)*, Santa Clara, California, June 2024. IEEE.

Debajyoti Halder, Manas Acharya, Aniket Malsane, Anshul Gandhi, and Erez Zadok. Empirical evaluation of ML models for per-job power prediction. In *Proceedings of the 7th ICPE Workshop on Hot Topics in Cloud Computing Performance (HotCloudPerf 2024)*, pages 181–188, London, UK, May 2024. ACM.

Yifei Liu, Manish Adkar, Gerard Holzmann, Geoff Kuenning, Pei Liu, Scott Smolka, Wei Su, and Erez Zadok. Metis: File system model checking via versatile input and state exploration. In *Proceedings of the 22nd USENIX Conference on File and Storage Technologies (FAST '24)*, pages 123–140, Santa Clara, CA, February 2024. USENIX Association. Received all 3 Artifact-Evaluation badges.

Tyler Estro, Mario Antunes, Pranav Bhandari, Anshul Gandhi, Geoff Kuenning, Yifei Liu, Carl Waldspurger, Avani Wildani, and Erez Zadok. Guiding simulations of multi-tier storage caches using knee detection. In *31st Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS '23)*, Stony Brook, NY, October 2023. IEEE Computer Society.

Peter Desnoyers, Andy Rudoff, Ian Adams, Tyler Estro, Anshul Gandhi, Geoff Kuenning, Mike Mesnier, Carl Waldspurger, Avani Wildani, and Erez Zadok. Persistent memory research in the post-optane era. In *1st Workshop on Disruptive Memory Systems (DIMES 2023)*, October 2023. https://dl.acm.org/doi/10.1145/3609308.3625268.

Docker EX1003
Page 248 of 310

Yifei Liu, Gautam Ahuja, Geoff Kuenning, Scott Smolka, and Erez Zadok. Input and output coverage needed in file system testing. In *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '23)*, Boston, MA, July 2023. ACM.

Alex Merenstein, Vasily Tarasov, Ali Anwar, Scott Guthridge, and Erez Zadok. F3: Serving files efficiently in serverless computing. In *Proceedings of the 16th ACM International Systems and Storage Conference (SYSTOR '23)*, Haifa, Israel, June 2023. ACM. Won Best Paper Award.

Ibrahim "Umit" Akgun, Santiago Vargas, Michael Arkhangelskiy, Andrew Burford, Michael McNeill, Aruna Balasubramanian, Anshul Gandhi, and Erez Zadok. Predicting network buffer capacity for BBR fairness. In *36th Conference on Neural Information Processing Systems (NeurIPS 2022) Workshop on ML for Systems*, December 2022.

Anjul Tyagi, Tyler Estro, Geoff Kuenning, Erez Zadok, and Klaus Mueller. PC-Expo: A metrics-based interactive axes reordering method for parallel coordinate displays. *IEEE Transactions on Visualization and Computer Graphics*, October 2022.

Anshul Gandhi, Kanad Ghose, Kartik Gopalan, Syed Hussain, Dongyoon Lee, David Liu, Zhenhua Liu, Patrick McDaniel, Shuai Mu, and Erez Zadok. Metrics for sustainability in data centers. In *Proceedings of the 1st Workshop on Sustainable Computer Systems Design and Implementation (HotCarbon'22)*, San Diego, CA, USA, July 2022. USENIX.

Sayontan Ghosh, Amanpreet Singh, Meyer "Alex" Merenstein, Wei Su, Scott A. Smolka, Erez Zadok, and Niranjan Balasubramanian. SpecNFS: A challenge dataset towards extracting formal models from natural language specifications. In *Proceedings of the 13th Language Resources and Evaluation Conference (LREC'22)*, pages 2166–2176, Marseille, France, June 2022.

Wei Su, Yifei Liu, Gomathi Ganesan, Gerard Holzmann, Scott Smolka, Erez Zadok, and Geoff Kuenning. Model-checking support for file system development. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '21)*, pages 103–110, Virtual, July 2021. ACM.

Ibrahim 'Umit' Akgun, Ali Selman Aydin, Aadil Shaikh, Lukas Velikov, and Erez Zadok. A machine learning framework to improve storage system performance. In *Proceedings of the 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage '21)*, Virtual, July 2021. ACM.

Alex Merenstein, Vasily Tarasov, Ali Anwar, Deepavali Bhagwat, Julie Lee, Lukas Rupprecht, Dimitris Skourtis, Yang Yang, and Erez Zadok. CNSBench: A cloud native storage benchmark native storage. In *Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST '21)*, Virtual, February 2021. USENIX Association.

Alex Merenstein, Vasily Tarasov, Ali Anwar, Deepavali Bhagwat, Lukas Rupprecht, Dimitris Skourtis, and Erez Zadok. The case for benchmarking control operations in cloud native storage. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage (HotStorage '20)*, Boston, MA, July 2020. USENIX.

Tyler Estro, Pranav Bhandari, Avani Wildani, and Erez Zadok. Desperately seeking ... optimal multi-tier cache configurations. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Storage (HotStorage '20)*, Boston, MA, July 2020. USENIX.

Wei Su, Akshay Aurora, Ming Chen, and Erez Zadok. Supporting transactions for bulk NFSv4 compounds. In *Proceedings of the 13th ACM International Systems and Storage Conference (SYSTOR '20)*, Haifa, Israel, June 2020. ACM.

Ibrahim Umit Akgun, Geoff Kuenning, and Erez Zadok. Re-animator: Versatile high-fidelity storage-system tracing and replaying. In *Proceedings of the 13th ACM International Systems and Storage Conference (SYSTOR '20)*, pages 61–74, Haifa, Israel, June 2020. ACM.

Elinor Schoenfeld, Fan Ye, Bing Zhou, Zongxing Xie, Erez Zadok, Xi Cheng, Jacqueline B. Mondros, Patricia Bruckenthal, and Jignesh Patel. Smart aging: Utilizing low-cost, non-wearable and secure sensing technologies to support the health and wellbeing of community dwelling older adults. In *Proceedings of the HIMSS 20 Preconference Session on Aging and Tech Forum: Addressing the Silver Tsunami*, Orlando, FL, March 2020.

Ibrahim Umit Akgun, Ali Selman Aydin, and Erez Zadok. KMLib: Towards machine learning for operating systems. In *Proceedings of the 2020 On-Device Intelligence Workshop, co-located with the MLSys Conference*, February 2020. Refereed abstract+poster.

Zhen Cao, Geoff Kuenning, and Erez Zadok. Carver: Finding important parameters for storage system tuning. In *Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST '20)*, Santa Clara, CA, February 2020. USENIX Association.

Muhammad Wajahat, Aditya Yele, Tyler Estro, Anshul Gandhi, and Erez Zadok. Distribution fitting and performance modeling for storage traces. In *27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2019)*, Rennes, France, October 2019. IEEE Computer Society. (**Won best student paper award**).

Anjul Tyagi, Zhen Cao, Tyler Estro, Klaus Mueller, and Erez Zadok. ICE: Interactive configuration explorer for high dimensional categorical parameter spaces. In *IEEE Conference on Visual Analytics Science and Technology (VAST 2019)*, October 2019.

Zhen Cao, Geoff Kuenning, Klaus Mueller, Anjul Tyagi, and Erez Zadok. Graphs are not enough: Using interactive visual analytics in storage research. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Storage (HotStorage '19)*, Renton, WA, July 2019. USENIX.

Ming Chen and Erez Zadok. Kurma: Secure geo-distributed multi-cloud storage gateways. In *Proceedings of the 12th ACM International Systems and Storage Conference (SYSTOR '19)*, Haifa, Israel, June 2019. ACM.

Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In *Proceedings of the Annual USENIX Technical Conference*, Boston, MA, July 2018. USENIX Association. Data set at `http://download.filesystems.org/auto-tune/ATC-2018-auto-tune-data.sql.gz`.

S. Archak, S. Dixit, R.P. Spillane, and E. Zadok. Multi-tier caching, May 2018. US Patent 9,959,279.

Erez Zadok, Dean Hildebrand, Geoff Kuenning, and Keith Smith. POSIX is dead! Long live... errr... what exactly? In *Proceedings of the 9th USENIX Workshop on Hot Topics in Storage (HotStorage '17)*, Santa Clara, CA, July 2017. USENIX. WACI track.

Ming Chen, Dean Hildebrand, Henry Nelson, Jasmit Saluja, Ashok Subramony, and Erez Zadok. vNFS: Maximizing NFS performance with compounds and vectorized I/O. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*, pages 301–314, Santa Clara, CA, February-March 2017. USENIX Association. **(Nominated for best paper award)**.

Bharath Kumar Reddy Vangoor, Vasily Tarasov, and Erez Zadok. To FUSE or not to FUSE: Performance of user-space file systems. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*, pages 59–72, Santa Clara, CA, February-March 2017. USENIX Association.

Docker EX1003
Page 250 of 310

Zhen Cao, Vasily Tarasov, Hari Raman, Dean Hildebrand, and Erez Zadok. On the performance variation in modern storage stacks. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST '17)*, pages 329–343, Santa Clara, CA, February-March 2017. USENIX Association.

Ming Chen, Arun Vasudevan, Kelong Wang, and Erez Zadok. SeMiNAS: A secure middleware for wide-area network-attached storage. In *Proceedings of the 9th ACM International Systems and Storage Conference (ACM SYSTOR '16)*, Haifa, Israel, June 2016. ACM.

S. Archak, S. Dixit, R.P. Spillane, and E. Zadok. Multi-tier caching, May 2016. US Patent 9,355,109.

Zhen Sun, Geoff Kuenning, Sonam Mandal, Philip Shilane, Vasily Tarasov, Nong Xiao, and Erez Zadok. A long-term user-centric analysis of deduplication patterns. In *Proceedings of the 32nd International IEEE Symposium on Mass Storage Systems and Technologies (MSST '16)*, pages 1–7, Santa Clara, California, May 2016. IEEE.

Sonam Mandal, Geoff Kuenning, Dongju Ok, Varun Shastry, Pilip Shilane, Sun Zhen, Vasily Tarasov, and Erez Zadok. Using hints to improve inline block-layer deduplication. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST '16)*, pages 315–322, Santa Clara, CA, February 2016. USENIX Association.

Erez Zadok, Aashray Arora, Zhen Cao, Akhilesh Chaganti, Arvind Chaudhary, and Sonam Mandal. Parametric optimization of storage systems. In *HotStorage '15: Proceedings of the 7th USENIX Workshop on Hot Topics in Storage*, Santa Clara, CA, July 2015. USENIX, USENIX.

Vasily Tarasov, Abhishek Gupta, Kumar Sourav, Sagar Trehan, and Erez Zadok. Terra incognita: On the practicality of user-space file systems. In *HotStorage '15: Proceedings of the 7th USENIX Workshop on Hot Topics in Storage*, Santa Clara, CA, July 2015. USENIX, USENIX.

Ming Chen, Dean Hildebrand, Geoff Kuenning, Soujanya Shankaranarayana, Bharat Singh, and Erez Zadok. Newer is sometimes better: An evaluation of NFSv4.1. In *Proceedings of the 2015 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2015)*, Portland, OR, June 2015. ACM.

M. Chen, D. Hildebrand, G. Kuenning, S. Shankaranarayana, V. Tarasov, A. Vasudevan, E. Zadok, and K. Zakirova. Linux NFSv4.1 performance under a microscope. In *Proceedings of USENIX LISA*. USENIX Association, November 2014. Extended Abstract.

Vasily Tarasov, Deepak Jain, Geoff Kuenning, Sonam Mandal, Karthikeyani Palanisami, Philip Shilane, Sagar Trehan, and Erez Zadok. Dmdedup: Device mapper target for data deduplication. In *Proceedings of the Linux Symposium*, pages 83–95, Ottawa, Canada, July 2014.

Z. Li, A. Mukker, and E. Zadok. On the importance of evaluating storage systems' $costs. In *Proceedings of the 6th USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'14, 2014.

Vasily Tarasov, Deepak Jain, Dean Hildebrand, Renu Tewari, Geoff Kuenning, and Erez Zadok. Improving I/O performance using virtual disk introspection. In *Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, June 2013.

Vasily Tarasov, Dean Hildebrand, Geoff Kuenning, and Erez Zadok. Virtual machine workloads: The case for new benchmarks for NAS. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '13)*, San Jose, CA, February 2013. USENIX Association.

Pradeep Shetty, Richard Spillane, Ravikant Malpani, Binesh Andrews, Justin Seyster, and Erez Zadok. Building workload-independent storage with VT-trees. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '13)*, pages 17–30, San Jose, CA, February 2013. USENIX Association.

Z. Li, A. Desai, C. Bhatt, and E. Zadok. vATM: vSphere adaptive task management. In *In Proceedings of the Seventh International Workshop on Feedback Computing (FC'12)*, September 2012.

David S. H. Rosenthal, Daniel C. Rosenthal, Ethan L. Miller, Ian F. Adams, Mark W. Storer, and Erez Zadok. The economics of long-term digital storage. In *The Memory of the World in the Digital age: Digitization and Preservation*. United Nations Educational, Scientific and Cultural Organization (UNESCO), September 2012.

E. Bartocci, R. Grosu, A. Karmarkar, S. A. Smolka, S. D. Stoller, E. Zadok, and J. Seyster. Adaptive runtime verification. In *Proc. 3rd International Conference on Runtime Verification (RV'12)*, Istanbul, Turkey, September 2012.

Michael A. Bender, Martin Farach-Colton, Rob Johnson, Russell Kraner, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. In *Proceedings of the 38th International Conference on Very Large Data Bases (VLDB)*, Istanbul, Turkey, August 2012. Morgan Kaufmann.

Vasily Tarasov, Gyumin Sim, Anna Povzner, and Erez Zadok. Efficient I/O scheduling with accurately estimated disk drive latencies. In *In Proceedings of the 8th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT'12)*, 2012.

V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the Annual USENIX Technical Conference*, Boston, MA, June 2012. USENIX Association.

Z. Li, K. M. Greenan, A. W. Leung, and E. Zadok. Power consumption in enterprise-scale backup storage systems. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA, February 2012. USENIX Association.

V. Tarasov, K. S. Kumar, J. Ma, D. Hildebrand, A. Povzner, G. Kuenning, and E. Zadok. Extracting flexible, replayable models from large block traces. In *Proceedings of the Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA, February 2012. USENIX Association.

M.G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya, and S.S. J. Systems and methods for detection of new malicious executables, July 2011. US Patent 7,979,907.

Richard P. Spillane, Pradeep J. Shetty, Erez Zadok, Shrikar Archak, and Sagar Dixit. An efficient multi-tier tablet server storage architecture. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*, Cascais, Portugal, October 2011.

J. Seyster, P. Radhakrishnan, S. Katoch, A. Duggal, S. D. Stoller, and E. Zadok. Redflag: A framework for analysis of kernel-level concurrency. In *Proc. of the 11th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'11)*, Melbourne, Australia, October 2011.

S. D. Stoller, E. Bartocci, J. Seyster, R. Grosu, K. Havelund, S. A. Smolka, and E. Zadok. Runtime verification with state estimation. In *Proc. 2nd International Conference on Runtime Verification (RV'11)*, San Francisco, CA, September 2011. (**Won best paper award**).

Z. Li, R. Grosu, K. Muppalla, S. A. Smolka, S. D. Stoller, and E. Zadok. Model discovery for energy-aware computing systems: An experimental evaluation. In *Proceedings of the 1st Workshop on Energy Consumption and Reliability of Storage Systems (ERSS'11)*, Orlando, FL, July 2011.

Michel A. Bender, Martin Farach-Colton, Rob Johnson, Bradley C. Kuszmaul, Dzejla Medjedovic, Pablo Montes, Pradeep Shetty, Richard P. Spillane, and Erez Zadok. Don't thrash: How to cache your hash on flash. In *HotStorage '11: Proceedings of the 3rd USENIX Workshop on Hot Topics in Storage*, Portland, OR, June 2011. USENIX Association.

Z. Li, R. Grosu, P. Sehgal, S. A. Smolka, S. D. Stoller, and E. Zadok. On the energy consumption and performance of systems software. In *Proceedings of the 4th Israeli Experimental Systems Conference (ACM SYSTOR '11)*, Haifa, Israel, May/June 2011. ACM.

Vasily Tarasov, Saumitra Bhanage, Erez Zadok, and Margo Seltzer. Benchmarking file system benchmarking: It *IS* rocket science. In *Proceedings of HotOS XIII:The 13th USENIX Workshop on Hot Topics in Operating Systems*, Napa, CA, May 2011.

J. Seyster, K. Dixit, X. Huang, R. Grosu, K. Havelund, S. A. Smolka, S. D. Stoller, and E. Zadok. Aspect-oriented instrumentation with GCC. In *Proc. of the 1st International Conference on Runtime Verification (RV 2010)*, Lecture Notes in Computer Science. Springer, November 2010.

R. Spillane, S. Dixit, S. Archak, S. Bhanage, and E. Zadok. Exporting kernel page caching for efficient user-level I/O. In *Proceedings of the International IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, Incline Village, Nevada, May 2010. IEEE.

Priya Sehgal, Vasily Tarasov, and Erez Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '10)*, pages 253–266, San Jose, CA, February 2010. USENIX Association.

Patrick McDaniel, Kevin Butler, Stephen Mclaughlin, Radu Sion, Erez Zadok, and Marianne Winslett. Towards a secure and efficient system for end-to-end provenance. In *Proceedings of the Second USENIX workshop on the Theory and Practice of Provenance (TAPP '10)*, San Jose, CA, February 2010. USENIX Association.

M.G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya, and S.J. Salvatore. System and methods for detection of new malicious executables, February 2009. US Patent 7,487,544.

D. J. Dean, S. Callanan, and E. Zadok. The visual development of GCC plug-ins. In *Proceedings of the 2009 GCC Developers' Summit*, Montreal, Canada, June 2009.

C. Yalamanchili, G. Sivathanu, K. Vijayasankar, and E. Zadok. DHIS: Discriminating hierarchical storage. In *Proceedings of the Second ACM Israeli Experimental Systems Conference (SYSTOR '09)*, Haifa, Israel, May 2009. ACM.

Rachita Kothiyal, Vasily Tarasov, Priya Sehgal, and Erez Zadok. Energy and performance evaluation of lossless file data compression on server systems. In *Proceedings of the Second ACM Israeli Experimental Systems Conference (SYSTOR '09)*, Haifa, Israel, May 2009. ACM.

R. Spillane, R. Sears, C. Yalamanchili, S. Gaikwad, M. Chinni, and E. Zadok. Story Book: An efficient extensible provenance framework. In *Proceedings of the First USENIX workshop on the Theory and Practice of Provenance (TAPP '09)*, San Francisco, CA, February 2009. USENIX Association.

R. P. Spillane, S. Gaikwad, E. Zadok, C. P. Wright, and M. Chinni. Enabling transactional file access via lightweight kernel extensions. In *Proceedings of the Seventh USENIX Conference on File*

and Storage Technologies (FAST '09), pages 29–42, San Francisco, CA, February 2009. USENIX Association.

S. Sundararaman, G. Sivathanu, and E. Zadok. Selective versioning in a secure disk system. In *Proceedings of the 17th USENIX Security Symposium*, pages 259–274, San Jose, CA, July-August 2008. USENIX Association.

A. Traeger, I. Deras, and E. Zadok. DARC: Dynamic analysis of root causes of latency distributions. In *Proceedings of the 2008 International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2008)*, pages 277–288, Annapolis, MD, June 2008. ACM.

D. P. Quigley, K. Albrecht, and E. Zadok. PLEASE: Policy language for easy administration of SELinux. In *Proceedings of the Second annual Computer Security Conference (CSC 2008)*, Myrtle Beach, SC, April 2008.

S. Callanan, D. J. Dean, M. Gorbovitski, R. Grosu, J. Seyster, S. A. Smolka, S. D. Stoller, and E. Zadok. Software monitoring with bounded overhead. In *Proceedings of the 2008 NSF Next Generation Software Workshop, in conjunction with the 2008 International Parallel and Distributed Processing Symposium (IPDPS 2008)*, Miami, FL, April 2008.

E. Zadok. The layers are coming, the layers are coming. In *The 2008 Linux Storage and Filesystem Workshop (LSF'08)*, San Jose, CA, February 2008. USENIX Association.

K. Vijayasankar, G. Sivathanu, S. Sundararaman, and E. Zadok. Exploiting type-awareness in a self-recovering disk. In *Proceedings of the Third ACM Workshop on Storage Security and Survivability (StorageSS 2007)*, pages 25–30, Alexandria, VA, October 2007. ACM.

A. Traeger, K. Thangavelu, and E. Zadok. Round-trip privacy with NFSv4. In *Proceedings of the Third ACM Workshop on Storage Security and Survivability (StorageSS 2007)*, pages 1–7, Alexandria, VA, October 2007. ACM.

N. Joukov, A. M. Krishnakumar, C. Patti, A. Rai, S. Satnur, A. Traeger, and E. Zadok. RAIF: Redundant array of independent filesystems. In *Proceedings of 24th IEEE Conference on Mass Storage Systems and Technologies (MSST 2007)*, pages 199–212, San Diego, CA, September 2007. IEEE.

S. Callanan, D. J. Dean, and E. Zadok. Extending GCC with modular GIMPLE optimizations. In *Proceedings of the 2007 GCC Developers' Summit*, pages 31–37, Ottawa, Canada, July 2007.

Richard P. Spillane, Charles P. Wright, Gopalan Sivathanu, and Erez Zadok. Rapid file system development using ptrace. In *Proceedings of the Workshop on Experimental Computer Science (ExpCS), in conjunction with ACM FCRC*, page 22, San Diego, CA, June 2007.

J. Sipek, Y. Pericleous, and E. Zadok. Kernel support for stackable file systems. In *Proceedings of the 2007 Ottawa Linux Symposium (OLS 2007)*, volume 2, pages 223–227, Ottawa, Canada, June 2007.

A. D. A. Martin and E. Zadok. Autofs - an automounting file system for freebsd 6.x. In *Proceedings of the 2007 BSDCan Technical BSD Conference*, Ottawa, ON, Canada, May 2007.

S. Callanan, R. Grosu, J. Seyster, S. A. Smolka, and E. Zadok. Model predictive control for memory profiling. In *Proceedings of the 2007 NSF Next Generation Software Workshop, in conjunction with the 2007 International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Long beach, CA, March 2007.

J. Sipek and E. Zadok. Making Linux stacking-friendly. In *The 2007 Linux Storage and Filesystem Workshop (LSF'07)*, San Jose, CA, February 2007. USENIX Association.

G. Sivathanu, S. Sundararaman, and E. Zadok. Type-safe disks. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 15–28, Seattle, WA, November 2006. ACM SIGOPS.

Nikolai Joukov, Ashivay Traeger, Rakesh Iyer, Charles P. Wright, and Erez Zadok. Operating system profiling via latency analysis. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, pages 89–102, Seattle, WA, November 2006. ACM SIGOPS.

N. Joukov, H. Papaxenopoulos, and E. Zadok. Secure deletion myths, issues, and solutions. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability (StorageSS 2006)*, pages 61–66, Alexandria, VA, October 2006. ACM.

A. Traeger, N. Joukov, J. Sipek, and E. Zadok. Using free web storage for data backup. In *Proceedings of the Second ACM Workshop on Storage Security and Survivability (StorageSS 2006)*, pages 73–77, Alexandria, VA, October 2006. ACM.

D. P. Quigley, J. Sipek, C. P. Wright, and E. Zadok. UnionFS: User- and community-oriented development of a unification filesystem. In *Proceedings of the 2006 Linux Symposium*, volume 2, pages 349–362, Ottawa, Canada, July 2006.

S. Callanan, R. Grosu, X. Huang, S. A. Smolka, and E. Zadok. Compiler-assisted software verification using plug-ins. In *Proceedings of the 2006 NSF Next Generation Software Workshop, in conjunction with the 2006 International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece, April 2006. DOI 10.1109/IPDPS.2006.1639579.

S. Callanan, R. Grosu, A. Rai, S. A. Smolka, M. R. True, and E. Zadok. Runtime verification for high-confidence systems: A Monte Carlo approach. In *Proceedings of the Second Workshop on Model Based Testing (MBT 2006), held in conjunction with the European Joint Conferences on Theory and Practice of Software (ETAPS 2006)*, pages 41–53, Vienna, Austria, March 2006.

Nikolai Joukov, Timothy Wong, and Erez Zadok. Accurate and efficient replaying of file system traces. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '05)*, pages 337–350, San Francisco, CA, December 2005. USENIX Association.

N. Joukov and E. Zadok. Adding secure deletion to your favorite file system. In *Proceedings of the third international IEEE Security In Storage Workshop (SISW 2005)*, pages 63–70, San Francisco, CA, December 2005. IEEE Computer Society.

N. Joukov, A. Kashyap, G. Sivathanu, and E. Zadok. Kefence: An electric fence for kernel buffers. In *Proceedings of the First ACM Workshop on Storage Security and Survivability (StorageSS 2005)*, pages 37–43, FairFax, VA, November 2005. ACM. (**Won best short paper award**).

G. Sivathanu, C. P. Wright, and E. Zadok. Ensuring data integrity in storage: Techniques and applications. In *Proceedings of the First ACM Workshop on Storage Security and Survivability (StorageSS 2005)*, pages 26–36, FairFax, VA, November 2005. ACM.

N. Joukov, A. Rai, and E. Zadok. Increasing distributed storage survivability with a stackable raid-like file system. In *Proceedings of the 2005 IEEE/ACM Workshop on Cluster Security, in conjunction with the Fifth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, pages 82–89, Cardiff, UK, May 2005. IEEE. (**Won best paper award**).

C. P. Wright, N. Joukov, D. Kulkarni, Y. Miretskiy, and E. Zadok. Auto-pilot: A platform for system software benchmarking. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 175–187, Anaheim, CA, April 2005. USENIX Association.

E. Zadok, S. Callanan, A. Rai, G. Sivathanu, and A. Traeger. Efficient and safe execution of user-level code in the kernel. In *Proceedings of the 2005 NSF Next Generation Software Workshop, in conjunction with the 2005 International Parallel and Distributed Processing Symposium (IPDPS 2005)*, page 221.1, Denver, CO, April 2005.

A. Kashyap, S. Patil, G. Sivathanu, and E. Zadok. I3FS: An in-kernel integrity checker and intrusion detection file system. In *Proceedings of the 18th USENIX Large Installation System Administration Conference (LISA 2004)*, pages 69–79, Atlanta, GA, November 2004. USENIX Association.

Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An on-access anti-virus file system. In *Proceedings of the 13th USENIX Security Symposium (Security 2004)*, pages 73–88, San Diego, CA, August 2004. USENIX Association.

K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok. A versatile and user-oriented versioning file system. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '04)*, pages 115–128, San Francisco, CA, March/April 2004. USENIX Association.

Akshat Aranya, Charles P. Wright, and Erez Zadok. Tracefs: A file system to trace them all. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST '04)*, pages 129–143, San Francisco, CA, March/April 2004. USENIX Association.

E. Zadok, J. Osborn, A. Shater, C. P. Wright, K. Muniswamy-Reddy, and J. Nieh. Reducing storage management costs via informed user-based policies. In *Proceedings of the 12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*, pages 193–197, College Park, MD, April 2004. IEEE.

C. P. Wright, J. Dave, and E. Zadok. Cryptographic file systems performance: What you don't know can hurt you. In *Proceedings of the Second IEEE International Security In Storage Workshop (SISW 2003)*, pages 47–61, Washington, DC, October 2003. IEEE Computer Society.

C. P. Wright, M. Martino, and E. Zadok. NCryptfs: A secure and convenient cryptographic file system. In *Proceedings of the Annual USENIX Technical Conference*, pages 197–210, San Antonio, TX, June 2003. USENIX Association.

Amit Purohit, Charles P. Wright, Joseph Spadavecchia, and Erez Zadok. Cosy: Develop in user-land, run in kernel-mode. In *Proceedings of the 2003 ACM Workshop on Hot Topics in Operating Systems (HotOS IX)*, pages 109–114, Lihue, Hawaii, May 2003. USENIX Association.

R. Grosu, E. Zadok, S. A. Smolka, R. Cleaveland, and Y. A. Liu. High-confidence operating systems. In *Proceedings of the Tenth SIGOPS European Workshop: "Can we really depend on an OS?"*, pages 205–208, Saint-Emilion, France, September 2002.

J. Spadavecchia and E. Zadok. Enhancing NFS cross-administrative domain access. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 181–194, Monterey, CA, June 2002. USENIX Association.

E. Zadok. Overhauling AMD for the '00s: A case study of GNU autotools. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 287–297, Monterey, CA, June 2002. USENIX Association.

E. Zadok, J. M. Anderson, I. Bădulescu, and J. Nieh. Fast indexing: Support for size-changing algorithms in stackable file systems. In *Proceedings of the Annual USENIX Technical Conference (ATC)*, pages 289–304, Boston, MA, June 2001. USENIX Association.

E. Zadok and J. Nieh. FiST: A language for stackable file systems. In *Proceedings of the Annual USENIX Technical Conference*, pages 55–70, San Diego, CA, June 2000. USENIX Association.

E. Zadok, I. Bădulescu, and A. Shender. Extending file systems using stackable templates. In *Proceedings of the Annual USENIX Technical Conference*, pages 57–70, Monterey, CA, June 1999. USENIX Association.

E. Zadok and I. Bădulescu. A stackable file system interface for Linux. In *LinuxExpo Conference Proceedings*, pages 141–151, Raleigh, NC, May 1999.

E. Zadok and A. Dupuy. HLFSD: Delivering Email to your $HOME. In *Proceedings of the Seventh USENIX Systems Administration Conference (LISA VII)*, pages 243–254, Monterey, CA, November 1993. USENIX Association.

E. Zadok and D. Duchamp. Discovery and hot replacement of replicated read-only file systems, with application to mobile computing. In *Proceedings of the Summer USENIX Technical Conference*, pages 69–85, Cincinnati, OH, June 1992. USENIX Association.

M. G. Schultz, E. Eskin, E. Zadok, M. Bhattacharyya, and S. J. Stolfo. MEF: Malicious Email Filter — A UNIX Mail Filter that Detects Malicious Windows Executables. In *Proceedings of the Annual USENIX Technical Conference, FREENIX Track*, pages 245–252, Boston, MA, June 2001. (**Won best student paper award**).

M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, Oakland, CA, May 2001.

W. Lee, W. Fan, M. Miller, S. Stolfo, and E. Zadok. Toward cost-sensitive modeling for intrusion detection and response. In *Workshop on Intrusion Detection and Prevention, Seventh ACM Conference on Computer Security (CCS)*, pages 1–22, Athens, Greece, November 2000.

Other Publications

Maliha Tabassum. Integrity protections for secure long-term storage. Technical Report FSL-24-05, Computer Science Department, Stony Brook University, December 2024.

Yifei Liu. Towards efficient, scalable, and versatile file system model checking. Technical Report FSL-24-04, Computer Science Department, Stony Brook University, November 2024.

Christopher Smith. Long-term secure archival using proactively secret-shared datastores. Technical Report FSL-24-03, Computer Science Department, Stony Brook University, September 2024.

Alex Merenstein. *Techniques for Storage Performance Measurement and Data Management in Container-Native and Serverless Environments*. PhD thesis, May 2024.

Christopher Smith, Maliha Tabassum, Soumya Chowdary Daruru, Gaurav Kulhare, Arvin Wang, Ethan Miller, and Erez Zadok. Plugging the leaks in secure archival systems. Non-archival Poster presentation at Stony Brook University Department of Computer Science Graduate Research Day, March 2024.

Christopher Smith, Maliha Tabassum, Soumya Chowdary Daruru, Gaurav Kulhare, Arvin Wang, Ethan Miller, and Erez Zadok. Plugging the leaks in secure archival systems. Work in Progress presentation at the 22nd USENIX Conference on File and Storage Technologies (FAST'24), February 2024.

Docker EX1003
Page 257 of 310

Christopher Smith, Maliha Tabassum, Soumya Chowdary Daruru, Gaurav Kulhare, Arvin Wang, Ethan Miller, and Erez Zadok. Plugging the leaks in secure archival systems. Non-archival Poster presentation in the 22nd USENIX Conference on File and Storage Technologies (FAST'24), February 2024.

Alex Merenstein. Moving beyond host based virtualization: New techniques for performance measurement and data management in cloud native environments. Technical Report FSL-24-01, Computer Science Department, Stony Brook University, January 2024.

Mario Antunes, Tyler Estro, Pranav Bhandari, Anshul Gandhi, Geoff Kuenning, Yifei Liu, Carl Waldspurger, Avani Wildani, and Erez Zadok. (multi)knee/elbow point detection library, December 2023.

Syed Rafiul Hussain, Patrick McDaniel, Anshul Gandhi, Kanad Ghose, Kartik Gopalan, Dongyoon Lee, Yu David Liu, Zhenhua Liu, Shuai Mu, and Erez Zadok. Verifiable sustainability in data centers, July 2023. https://arxiv.org/abs/2307.11993.

Tyler Estro, Mario Antunes, Pranav Bhandari, Anshul Gandhi, Geoff Kuenning, Yifei Liu, Carl Waldspurger, Avani Wildani, and Erez Zadok. Guiding simulations of multi-tier storage caches using knee detection. Technical Report FSL-23-01, Computer Science Department, Stony Brook University, July 2023.

Elinor Schoenfeld, Tracy Trimboli, Givenchy Ayisi-Boahene, Patricia Bruckenthal, Shelley Horwitz, Erez Zadok, and Fan Ye. Smart aging: Engaging older adults to guide sensor development to support aging in place. In *Issues and experiences of integrating patient and community stakeholder perspectives in aging research*, volume 7, pages 583–583, Tampa, FL, Dec 2023.

Elinor Schoenfeld, Tracy Trimboli, Kaylyn Finnerty, Givenchy Ayisi-Boahene, Patricia Bruckenthal, Shelley Horwitz, Erez Zadok, and Fan Ye. Engaging older adults to guide low-cost non-wearable sensor technology development to age in place: Survey findings. In *Gerontology Society of America Annual Meeting (GSA 2023)*, volume 7, pages 692–692, Tampa, FL, Dec 2023.

Yifei Liu, Gerard Holzmann, Geoff Kuenning, Scott Smolka, and Erez Zadok. The case for model checking emerging file systems. Non-archival poster presentation in the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI '23), July 2023.

Tyler Estro, Kamalnath Polakam, and Erez Zadok. Using bash notebooks for systems education (user meeting 23). `https://www.chameleoncloud.org/experiment/share/82fbba82-1b34-4562-9866-b7adaec94b19`.

Ibrahim "Umit" Akgun. *Using Machine Learning to Improve Operating Systems' I/O Subsystems*. PhD thesis, December 2022.

Ibrahim "Umit" Akgun. Using machine learning to improve operating systems' I/O subsystems. Technical Report FSL-22-02, Computer Science Department, Stony Brook University, June 2022.

Sayontan Ghosh, Amanpreet Singh, Alex Merenstein, Wei Su, Scott Smolka, Erez Zadok, and Niranjan Balasubramanian. SpecNFS: A challenge dataset towards extracting formal models from natural language specifications. Non-archival poster presentation in the Workshop on Natural Language Processing for Programming in the ACL'21 conference, August 2021.

Tyler Estro. Accelerating multi-tier cache evaluations with intelligent MRC point selection. Technical Report FSL-21-01, Computer Science Department, Stony Brook University, January 2021.

Wei Su. Supporting transactions for bulk NFSv4 compounds. Technical Report FSL-20-02, Computer Science Department, Stony Brook University, December 2020.

Alex Merenstein. CNSBench: A cloud native storage benchmark. Technical Report FSL-20-01, Computer Science Department, Stony Brook University, December 2020.

Ibrahim "Umit" Akgun. Re-animator: Versatile high-fidelity system-call tracing and replaying. Technical Report FSL-19-02, Computer Science Department, Stony Brook University, May 2019.

George Amvrosiadis, Ali R. Butt, Vasily Tarasov, Erez Zadok, Ming Zhao, Irfan Ahmad, Remzi H. Arpaci-Dusseau, Feng Chen, Yiran Chen, Yong Chen, Yue Cheng, Vijay Chidambaram, Dilma Da Silva, Angela Demke-Brown, Peter Desnoyers, Jason Flinn, Xubin He, Song Jiang, Geoff Kuenning, Min Li, Carlos Maltzahn, Ethan L. Miller, Kathryn Mohror, Raju Rangaswami, Narasimha Reddy, David Rosenthal, Ali Saman Tosun, Nisha Talagala, Peter Varman, Sudharshan Vazhkudai, Avani Waldani, Xiaodong Zhang, Yiying Zhang, and Mai Zheng. Data storage research vision 2025: Report on NSF visioning workshop held may 30–june 1, 2018. Technical report, National Science Foundation, February 2019. https://dl.acm.org/citation.cfm?id=3316807.

Zhen Cao. *A Practical Auto-Tuning Framework for Storage Systems*. PhD thesis, Stony Brook University, January 2019.

Zhen Cao. A practical, real-time auto-tuning framework for storage systems. Technical Report FSL-18-01, Computer Science Department, Stony Brook University, April 2018.

M. Chen. *Kurma: Efficient and Secure Multi-Cloud Storage Gateways for Network-Attached Storage*. PhD thesis, Computer Science Department, Stony Brook University, April 2017. Technical Report FSL-17-01.

Bharath Kumar Reddy Vangoor. To FUSE or not to FUSE? analysis and performance characterization of the FUSE user-space file system framework. Master's thesis, Stony Brook University, December 2016. Technical Report FSL-16-02, http://www.fsl.cs.sunysb.edu/docs/fuse/bharath-msthesis.pdf.

Zhen Cao. Parametric optimization of storage systems. Technical Report FSL-16-01, Computer Science Department, Stony Brook University, January 2016.

Sonam Mandal. Design and implementation of an open-source deduplication platform for research. Technical Report FSL-15-03, Computer Science Department, Stony Brook University, December 2015.

Ming Chen. Kurma: Geo-distributed secure middlewares for cloud-backed network attached storage. Technical Report FSL-15-02, Computer Science Department, Stony Brook University, November 2015.

Arun Olappamanna Vasudevan. Finding the right balance: Security vs. performance with network storage systems. Master's thesis, Stony Brook University, May 2015. Technical Report FSL-15-01, http://www.fsl.cs.sunysb.edu/docs/arun-msthesis/arun-msthesis.pdf.

Benixon Arul Dhas, Erez Zadok, James Borden, and Jim Malina. Evaluation of Nilfs2 for shingled magnetic recording (SMR) disks. Technical Report FSL-14-03, Stony Brook University, September 2014.

M. Chen, D. Hildebrand, G. Kuenning, S. Shankaranarayana, V. Tarasov, A. Vasudevan, E. Zadok, and K. Zakirova. Linux NFSv4.1 performance under a microscope. Technical Report FSL-14-02, Stony Brook University, August 2014.

Z. Li. *GreenDM: A Versatile Tiering Hybrid Drive for the Trade-Off Evaluation of Performance, Energy, and Endurance*. PhD thesis, Computer Science Department, Stony Brook University, May 2014.

M. Palmur, Z. Li, and E. Zadok. Cpuidle from user space. Technical Report FSL-13-05, Computer Science Department, Stony Brook University, December 2013.

V. Tarasov. *Multi-dimensional Workload Analysis and Synthesis for Modern Storage Systems*. PhD thesis, Computer Science Department, Stony Brook University, December 2013.

Zhichao Li. System-aware resource scheduling for performance, energy, and reliability in tiered storage systems. Technical Report FSL-13-03, Computer Science Department, Stony Brook University, May 2013.

V. Tarasov. Multi-dimensional workload analysis and synthesis for modern storage systems. Technical Report FSL-13-02, Computer Science Department, Stony Brook University, April 2013.

M. Chen and E. Zadok. Greendm: A versatile hybrid drive for energy and performance. Technical Report FSL-13-01, Computer Science Department, Stony Brook University, April 2013.

J. Seyster. *Runtime Verification of Kernel-Level Concurrency Using Compiler-Based Instrumentation*. PhD thesis, Computer Science Department, Stony Brook University, December 2012. Technical Report FSL-12-05, `www.fsl.cs.sunysb.edu/docs/jseyster-dissertation/redflag.pdf`.

Pradeep J. Shetty, Richard P. Spillane, Ravikant R. Malpani, Binesh Andrews, Justin Seyster, and Erez Zadok. Unifying file systems and databases efficiently. Technical Report FSL-12-06, Stony Brook University Computer Science, September 2012.

A. Mudrankit. A context aware block layer: The case for block layer deduplication. Master's thesis, Stony Brook University, May 2012. Technical Report FSL-12-04.

P. J. Shetty. From tuples to files: a fast transactional system store and file system. Master's thesis, Stony Brook University, May 2012. Technical Report FSL-12-03.

S. Archak, S. Dixit, R.P. Spillane, and E. Zadok. Multi-tier caching, March 2012. US Patent App. 13/159,039.

R. P. Spillane. *Efficient, Scalable, and Versatile Application and System Transaction Management for Direct Storage Layers*. PhD thesis, Computer Science Department, Stony Brook University, Jan 2012. Technical Report FSL-12-02.

J. Seyster. Runtime verification of kernel-level concurrency using compiler-based instrumentation. Technical Report FSL-12-01, Computer Science Department, Stony Brook University, January 2012. `www.fsl.cs.sunysb.edu/docs/jseyster-proposal/redflag.pdf`.

A. Duggal. Stopping data races using Redflag. Master's thesis, Stony Brook University, May 2010. Technical Report FSL-10-02, `http://www.fsl.cs.sunysb.edu/docs/abhinav-thesis/abhinav_thesis.pdf`.

P. Sehgal. Optimizing energy and performance for server-class file system workloads. Master's thesis, Stony Brook University, May 2010. Technical Report FSL-10-01.

S. Callanan. *Flexible Debugging with Controllable Overhead*. PhD thesis, Computer Science Department, Stony Brook University, August 2009. Technical Report FSL-09-01, `http://www.fsl.cs.sunysb.edu/docs/callanan09phdthesis/callanan09phdthesis.pdf`.

R. P. Spillane. SAMT performance proof. Technical Report FSL-09-03, Stony Brook University Computer Science, June 2009. www.fsl.cs.sunysb.edu/~rick.

D. Dean. The visual development of GCC plug-ins with GDE. Master's thesis, Stony Brook University, April 2009. Technical Report FSL-09-04.

R. Kothiyal. Energy and performance evaluation of lossless file data compression on computer systems. Master's thesis, Stony Brook University, April 2009. Technical Report FSL-09-02.

R. P. Spillane, S. Gaikwad, E. Zadok, C. P. Wright, and M. Chinni. Enabling transactional file access via lightweight kernel extensions. Technical Report FSL-08-05, Stony Brook University Computer Science, August 2008.

A. Traeger. *Analyzing Root Causes of Latency Distributions*. PhD thesis, Computer Science Department, Stony Brook University, August 2008. Technical Report FSL-08-04.

J. Seyster. Techniques for visualizing software execution. Technical Report FSL-08-03, Computer Science Department, Stony Brook University, March 2008. www.fsl.cs.sunysb.edu/docs/debugging-tr/software-visualization-tr.pdf.

S. Callanan. Remote debugging with controllable overhead. Technical Report FSL-08-02, Computer Science Department, Stony Brook University, March 2008. www.fsl.cs.sunysb.edu/docs/debugging-tr/debugging-proposal.pdf.

G. Sivathanu. *End-to-End Abstractions for Application-Aware Storage*. PhD thesis, Computer Science Department, Stony Brook University, May 2008. Technical Report FSL-08-01.

D. P. Quigley. PLEASE: Policy language for easy administration of SELinux. Master's thesis, Stony Brook University, May 2007. Technical Report FSL-07-02, www.fsl.cs.sunysb.edu/docs/dquigley-msthesis/dquigley-msthesis.pdf.

N. Joukov. *Versatile, Portable, and Efficient File System Profiling*. PhD thesis, Computer Science Department, Stony Brook University, December 2006. Technical Report FSL-06-05.

C. P. Wright. *Extending ACID Semantics to the File System via ptrace*. PhD thesis, Computer Science Department, Stony Brook University, May 2006. Technical Report FSL-06-04.

N. Joukov, A. Traeger, R. Iyer, C. P. Wright, and E. Zadok. Latency as a viable metric for OS profiling. Technical Report FSL-06-03, Computer Science Department, Stony Brook University, April 2006.

R. Spillane, C. P. Wright, G. Sivathanu, and E. Zadok. Rapid file system development using ptrace. Technical Report FSL-06-02, Computer Science Department, Stony Brook University, January 2006.

C. P. Wright, R. Spillane, G. Sivathanu, and E. Zadok. Extending ACID semantics to the file system. Technical Report FSL-06-01, Computer Science Department, Stony Brook University, January 2006. www.fsl.cs.sunysb.edu/docs/amino-tr/amino.pdf.

G. Sivathanu and E. Zadok. A versatile persistent caching framework for file systems. Technical Report FSL-05-05, Computer Science Department, Stony Brook University, December 2005. www.fsl.cs.sunysb.edu/docs/cachefs-tr/cachefs.pdf.

N. Joukov, A. Traeger, C. P. Wright, and E. Zadok. Benchmarking file system benchmarks. Technical Report FSL-05-04, Computer Science Department, Stony Brook University, December 2005. www.fsl.cs.sunysb.edu/docs/fsbench/fsbench.pdf.

Docker EX1003
Page 261 of 310

S. Satnur. Storage virtualization with a stackable file system. Master's thesis, Stony Brook University, December 2005. Technical Report FSL-05-03, www.fsl.cs.sunysb.edu/docs/sunil-msthesis/sunil-msthesis.pdf.

E. Zadok, S. Callanan, R. Grosu, A. Rai, S. Smolka, and M. R. True. Runtime verification for high-confidence OS kernels: A Monte Carlo approach. Technical Report FSL-05-02, Computer Science Department, Stony Brook University, November 2005.

A. Rai and E. Zadok. On the role of static analysis in operating system checking and runtime verification. Technical Report FSL-05-01, Computer Science Department, Stony Brook University, May 2005. www.fsl.cs.sunysb.edu/docs/kgcc-rpe/kgcc.pdf.

E. Zadok, C. Wright, A. Aranya, A. Das, Y. Miretskiy, K.K. Muniswamy-Reddy, and A. Himmer. Stackable file systems and methods thereof, December 2005. US Patent App. 10/862,212.

A. Kashyap. File system extensibility and reliability using an in-kernel database. Master's thesis, Stony Brook University, December 2004. Technical Report FSL-04-06, www.fsl.cs.sunysb.edu/docs/kbdbfs-msthesis/kbdbfs.pdf.

A. Aranya. Versatile file system tracing with Tracefs. Master's thesis, Stony Brook University, August 2004. Technical Report FSL-04-05, www.fsl.cs.sunysb.edu/docs/tracefs-msthesis/tracefs.pdf.

G. Sivathanu, C. P. Wright, and E. Zadok. Enhancing file system integrity through checksums. Technical Report FSL-04-04, Computer Science Department, Stony Brook University, May 2004. www.fsl.cs.sunysb.edu/docs/nc-checksum-tr/nc-checksum.pdf.

A. Traeger, A. Rai, C. P. Wright, and E. Zadok. NFS file handle security. Technical Report FSL-04-03, Computer Science Department, Stony Brook University, May 2004. www.fsl.cs.sunysb.edu/docs/nfscrack-tr/nfscrack.pdf.

C. P. Wright and E. Zadok. Operating system support for extensible secure file systems. Technical Report FSL-04-02, Computer Science Department, Stony Brook University, May 2004. www.fsl.cs.sunysb.edu/docs/secfs-rpe/secfs.pdf.

C. P. Wright, J. Dave, P. Gupta, H. Krishnan, E. Zadok, and M. N. Zubair. Versatility and Unix semantics in a fan-out unification file system. Technical Report FSL-04-01b, Computer Science Department, Stony Brook University, October 2004. www.fsl.cs.sunysb.edu/docs/unionfs-tr/unionfs.pdf.

K. Muniswamy-Reddy. Versionfs: A versatile and user-oriented versioning file system. Master's thesis, Stony Brook University, December 2003. Technical Report FSL-03-03, www.fsl.cs.sunysb.edu/docs/versionfs-msthesis/versionfs.pdf.

C. P. Wright, J. Dave, and E. Zadok. Cryptographic file systems performance: What you don't know can hurt you. Technical Report FSL-03-02, Computer Science Department, Stony Brook University, August 2003. www.fsl.cs.sunysb.edu/docs/nc-perf/perf.pdf.

E. Zadok, J. Osborn, A. Shater, C. P. Wright, K. Muniswamy-Reddy, and J. Nieh. Reducing storage management costs via informed user-based policies. Technical Report FSL-02-02, Computer Science Department, Stony Brook University, September 2002. www.fsl.cs.sunysb.edu/docs/equota-policy/policy.pdf.

Docker EX1003
Page 262 of 310

A. Purohit, J. Spadavecchia, C. Wright, and E. Zadok. Improving application performance through system call composition. Technical Report FSL-02-01, Computer Science Department, Stony Brook University, June 2003. www.fsl.cs.sunysb.edu/docs/cosy-perf/.

O. C. Leonard, J. Nieh, E. Zadok, J. Osborn, A. Shater, and C. Wright. The design and implementation of elastic quotas: A system for flexible file system management. Technical Report CUCS-014-02, Computer Science Department, Columbia University, June 2002. www.cs.columbia.edu/~library.

E. Zadok. *FiST: A System for Stackable File System Code Generation*. PhD thesis, Computer Science Department, Columbia University, May 2001. www.fsl.cs.sunysb.edu/docs/zadok-phd-thesis/thesis.pdf.

Wenke Lee, Matthew Miller, Salvatore Stolfo, Kahil Jallad, Christopher T Park, Erez Zadok, and Vijay Prabhakar. Toward cost-sensitive modeling for intrusion detection. Technical Report CUCS-002-00, Computer Science Department, Columbia University, 2000. http://academiccommons.columbia.edu/download/fedora_content/download/ac:110332/CONTENT/cucs-002-00.pdf.

Erez Zadok, Johan M Andersen, Ion Badulescu, and Jason Nieh. Performance of size-changing algorithms in stackable file systems. Technical Report CUCS-023-00, Computer Science Department, Columbia University, 2000. http://academiccommons.columbia.edu/download/fedora_content/download/ac:110385/CONTENT/cucs-023-00.pdf.

E. Zadok. Stackable file systems as a security tool. Technical Report CUCS-036-99, Computer Science Department, Columbia University, December 1999. www.cs.columbia.edu/~library.

E. Zadok and I. Bădulescu. Usenetfs: A stackable file system for large article directories. Technical Report CUCS-022-98, Computer Science Department, Columbia University, July 1998. www.cs.columbia.edu/~library.

E. Zadok, I. Bădulescu, and A. Shender. Cryptfs: A stackable vnode level encryption file system. Technical Report CUCS-021-98, Computer Science Department, Columbia University, June 1998. www.cs.columbia.edu/~library.

E. Zadok. *FiST: A File System Component Compiler*. PhD thesis, Computer Science Department, Columbia University, April 1997. www.fsl.cs.sunysb.edu/docs/zadok-thesis-proposal.

E. Zadok. Discovery and hot replacement of replicated read-only file systems, with application to mobile computing. Master's thesis, Computer Science Department, Columbia University, October 1997. Technical Report CUCS-036-94, www.cs.columbia.edu/~library.

E. Zadok and A. Lih. PGMAKE: A Portable Distributed Make System. Technical Report CUCS-035-94, Computer Science Department, Columbia University, July 1994. www.cs.columbia.edu/~library.

Manuals

C. P. Wright and E. Zadok. *The Auto-pilot Benchmarking Suite User Manual*, 2.1 edition, July 2005. www.filesystems.org/docs/auto-pilot/.

J. S. Pendry, N. Williams, and E. Zadok. *Am-utils User Manual*, 6.1b3 edition, July 2003. www.am-utils.org.

PATENTS

Multi-Tier Caching, U.S. Patent 9,959,279, issued May 1, 2018.

Multi-Tier Caching, U.S. Patent 9,355,109, issued May 31, 2016.

Systems and methods for detection of new malicious executables, U.S. Patent 7,979,907B2, issued July 12, 2011.

System and methods for detection of new malicious executables, U.S. Patent 7,487,544B2, issued February 3, 2009.

ACADEMIC SERVICE

Program Committee Chair

2019-2020	The 2020 USENIX Annual Technical Conference (ATC 2020) program committee co-chair.
2015	The 13th USENIX Conference on File and Storage Technologies (FAST 2015) program committee co-chair.
2012	The 5th Israeli Experimental Systems Conference (SYSTOR 2012), program committee co-chair.
2010	The First Usenix Workshop on Sustainable Information Technology (SustainIT 2010), program committee co-chair.
2008	The First File-systems and Storage Benchmarking workshop, program chair (hosted at the Storage Systems Research Center at the University of California at Santa Cruz).
2006	ACM Storage Security and Survivability (StorageSS) workshop, program committee co-chair.
2005	Usenix Annual Conference 2005, Invited Talks program co-chair
2004	Usenix Security 2004, Works-in-Progress program chair
2004	Usenix Security 2004, Poster presentations program chair
2003	USENIX FreeNIX 2003 Annual Conference, program chair

Panels

Oct 2021	NSF CAREER review panel
June 2013	Panelist, EMC University Day, "University/Industry Relationships"
Jan 2012	NSF Sustainability Research Network (SRN) program
May 2010	Department of Energy, Computational and Technology Research program
Feb 2009	Panel moderator, The First Usenix Workshop on Sustainable Information Technology (SustainIT 2010), "The Present and Future of Sustainability R&D"
Oct 2009	Panelist, Workshop on Hot Topics in Storage and File Systems (HotStorage'09), "Rethinking File Systems"
Oct 2009	NSF CAREER (CSR+CNS) review panel

Apr 2007	NSF Computer Systems Research (CSR) Parallel and Distributed Operating Systems (PDOS) review panel
Aug 2004	NSF CyberTrust PI meeting panelist, "Software Assurance," Pittsburgh, PA
Aug 2004	Workshop on Information Assurance Education, Stony Brook University (WIAED 2004), panel chair, "How to establish your own IA Curriculum"
Apr 2004	NSF Distributed Systems and Compilers (DSC) review panel
Apr 2001	NSF ITR Operating Systems and Compilers (OSC) review panel, small grants

Editor-in-Chief

Nov 2012–present	Editor-in-Chief, ACM Transactions on Storage (TOS). `https://dl.acm.org/journal/tos/`

Editor

2014	Editorial Board, USENIX Journal of Education in System Administration (JESA). `www.usenix.org/jesa/`
Aug 2009–2012	Associate Editor, ACM Transactions on Storage (TOS). `https://tos.acm.org/`
Jul 2001–Jan 2009	Bulletin Editor, Webmaster, and lists maintainer for IEEE Technical Committee on Operating Systems and Application Environments (TCOS). `www.tcos.org`.

Book Reviews

Summer 2014	Technical reviewer for *The Design and Implementation of the FreeBSD Operating System (2nd Edition)*, Addison-Wesley, 2014, by Marshall Kirk McKusick, George Neville-Neil, and Robert N.M. Watson.
Summer 2005	Technical reviewer for *Understanding the Linux Kernel*, O'Reilly, 3rd ed., by P. Bovet and M. Cesati
Summer 2005	Technical reviewer for *The Design and Implementation of the FreeBSD Operating System*, Addison-Wesley, 2005, by Marshall Kirk McKusick and George V. Neville-Neil (two chapters)
Summer 2002	Technical reviewer for *Understanding the Linux Kernel*, O'Reilly, 2nd ed., by P. Bovet and M. Cesati
January 1994	Chapter 14 in *UNIX Systems for Modern Architectures* by Curt Schimmel (Addison-Wesley, 1994)

Program Committee Member

2019	The 2019 USENIX Annual Technical Conference (ATC 2019), Renton, WA.
2018	The 10th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage '18), Boston, MA.
2018	The 9th Annual Non-Volatile Memories Workshop (NVME 2018), San Diego, CA.
2017	The 4th FAST/Linux Summit, Santa Clara, CA.
2016–2017	The 15th USENIX Conference on File and Storage Technologies (FAST '17), Santa Clara, CA.

2016	The 3rd FAST/Linux Summit, Santa Clara, CA.
2015	The 2nd FAST/Linux Summit, Santa Clara, CA.
2014	USENIX 2nd Summit for Educators in System Administration (SESA 2014), Seattle, Washington (co-located with LISA'14).
2014	The 7th Israeli ACM Experimental Systems Conference (ACM SYSTOR 2014), Haifa, Israel.
2014	The 12th USENIX Conference on File and Storage Technologies (FAST '14), Santa Clara, CA.
2014	The 1st FAST/Linux Summit, Santa Clara, CA.
2013	USENIX 1st Summit for Educators in System Administration (SESA 2013), Seattle, Washington (co-located with LISA'13).
2013	Interactions of NVM/Flash with Operating-Systems and Workloads Workshop (INFLOW 2013), Farmington, PA (co-located with SOSP'13).
2013	International Conference on Parallel and Distributed Systems (ICPADS 2013), Seoul, Korea.
2011	Operating System Support for Next Generation Large Scale NVRAM (NVRAMOS'11), Jeju Island, South Korea.
2011	The 4th Israeli Experimental Systems Conference (SYSTOR 2011), Haifa, Israel.
2011	The 27th IEEE Symposium on Massive Storage Systems and Technologies (MSST2011), Research Track. Colorado.
2011	The 2011 USENIX Annual Technical Conference
2010	The 26th IEEE Symposium on Massive Storage Systems and Technologies (MSST2010), Research Track. Incline Village, Nevada.
2010	The 3rd Israeli Experimental Systems Conference (SYSTOR 2010), Haifa, Israel.
2009	The 21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2009).
2009	The first Usenix workshop on the theory and practice of provenance (TAPP '09).
2009	The 2009 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2009), New York, NY.
2009	The 2nd Israeli Experimental Systems Conference (SYSTOR 2009), Haifa, Israel.
2008	Fifth USENIX Conference on File and Storage Technologies (FAST '08), San Jose, CA.
2007	The Third ACM International Workshop on Storage Security and Survivability (StorageSS 2007), Alexandria, VA
2007	Fifth USENIX Conference on File and Storage Technologies (FAST '07)
2006	CCGrid Cluster Security workshop (in conjunction with CCGrid 2006, the Sixth IEEE/ACM International Symposium on Cluster Computing and the Grid), Singapore, Korea.

2005	The 3rd International IEEE Security in Storage Workshop (co-sponsored by Usenix), San Francisco, CA
2005	The First ACM International Workshop on Storage Security and Survivability (StorageSS), "The Paradigm Shift to Info-Centric Protection," Fairfax, VA
2004–2005	CCGrid Cluster Security workshop (in conjunction with CCGrid 2005, the Fifth IEEE/ACM International Symposium on Cluster Computing and the Grid), Cardiff, UK
2004	Workshop on Information Assurance Education, Stony Brook University (WIAED 2004), organizer
2002	USENIX FreeNIX 2002 Annual Conference
2001	First USENIX Conference on File and Storage Technologies (FAST '02)

Session Chair

Oct 2020	Keynote by Richardo Biancini, the 13th ACM International System and Storage Conference
Jul 2019	Co-chair, ACM Workshop on on Hot Topics in Storage and File Systems (HotStorage), "File Systems"
Jan 2015	The annual NSF Secure and Trustworthy Computing (SaTC) program PI meeting (SaTC-PI 2015), "Benchmarking and workloads"
Feb 2014	The 12th USENIX Conference on File and Storage Technologies (FAST '14), "Performance and Efficiency"
June 2011	The 2011 Usenix Annual Technical Conference (Usenix ATC '11), "Storage Deduplication"
July 2013	The 6th Israeli Experimental Systems Conference (SYSTOR '13), "Flash Optimization"
May 2011	The 4th Israeli Experimental Systems Conference (ACM SYSTOR '11), "Adaptation and performance"
May 2009	The 2nd Israeli Experimental Systems Conference (ACM SYSTOR '09), "Deduplication"
Feb 2009	The first Usenix workshop on the theory and practice of provenance (TAPP '09)
May 2009	Fifth USENIX Conference on File and Storage Technologies (FAST '07), "Beyond the Machine Room"
Nov 2005	ACM International Workshop on Storage Security and Survivability (StorageSS), "Cryptographic Storage Security"
	Usenix Security 2004, Works-in-Progress session chair
Aug 2004	Usenix Security 2004, Works-in-Progress session chair
Aug 2004	Usenix Security 2004, Poster presentations session chair
Jun 2002	USENIX FreeNIX Annual Conference, "File Systems"

Docker EX1003
Page 267 of 310

Jun 2001	USENIX FreeNIX Annual Conference, “Scripting”

Referee

Feb 2022	USENIX FAST 2022
Nov 2020	ACM Transactions on Storage (TOS)
Oct 2018	IEEE Transactions on Parallel and Distributed Systems (TDPS)
Aug 2017	ACM Transactions on Storage (TOS)
2016	Runtime Verification (RV) 2016
2011	Sustainable Computing Informatics and Systems journal
Nov 2010	ACM SIGMOD 2010
Oct 2010	IEEE Internet Computing
Jan 2010	ACM Transactions on Computers (TOCS)
Dec 2009	IEEE Transactions on Parallel and Distributed Systems (TPDS)
Feb 2009	USENIX FAST 2009
Feb 2007	USENIX Annual conference 2007
Jan 2007	Sigmetrics 2007
Jul 2006	Elsevier’s Information Sciences journal
Feb 2006	USENIX Annual conference 2006
Aug 2005	USENIX FAST 2005
Aug 2004–present	ACM Transactions on Storage (TOS) journal
Jun 2004	USENIX OSDI 2004
Nov 2003	USENIX FAST 2004
Dec 2002	USENIX FREENIX 2003
Jun 2002	USENIX OSDI 2002
May 2002	ACM Mobile Networks and Applications (MONET)
Mar 2002	16th Annual ACM International Conference on Supercomputing (ICS)
Mar 2002	LCTES/SCOPES ’02: ACM SIGPLAN Joint Conference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems.
Jan 2002	USENIX FreeNIX Annual Conference
Mar 2001	8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)
Jan 2001	USENIX Annual Conference
Jan 2001	USENIX FreeNIX Annual Conference
May 2000	USENIX LISA 14
Dec 2000	USENIX Annual Conference

Docker EX1003
Page 268 of 310

May 1999	USENIX LISA 13
May 1999	ACM Twelfth Annual Symposium on User Interface Software and Technology (UIST '99)
Mar 1996	2nd International Mobile Computing Conference
1994	Trends in Software: Software Configuration Management

Committee Service

2021–present	Tiger Team lead, "Data Storage, Data Management, Distributed Ledger Technologies, Cybersecurity, and Biometrics," Stony Brook University.
2020–present	Steering Committee member, USENIX Annual Technical Conference (USENIX ATC)
2020–present	Steering Committee member, ACM Workshop on on Hot Topics in Storage and File Systems (HotStorage)
2021–2022	Co-Chair, President's "One Campus IT Operations" (OCITO) working group, Stony Brook University.
2017–2020	Member, Provost Committee on Faculty Mentorship, Stony Brook University
2017	Member, ad-hoc SL-2 hiring committee for CS Staff Assistant, Computer Science Department, Stony Brook University
2016–2020	Member, STRIDE project advisory committee, Institute for Advanced Computational Science (IACS) and Computer Science Department, Stony Brook University
2016–2019	Chair, Research and IT/Building Operations committee, Computer Science Department, Stony Brook University (note: also official mentor to all junior CS faculty).
2016-2017	Member, ad-hoc hiring committee for CEAS Grant Writer, College of Engineering and Applied Science, Stony Brook University
2016	Member, ABET funding cmte, College of Engineering and Applied Science, Stony Brook University
2015–present	Steering Committee member, USENIX Filesystem and Storage Technologies Conference (USENIX FAST)
2014	Member, Executive committee, Computer Science Department, Stony Brook University
2014	Member, ad-hoc committee for selection of departmental award, Computer Science Department, Stony Brook University
2014	Member, Provost's Outstanding Lecturer Selection Committee, Stony Brook University
2013	Member, Provost's Interdisciplinary Faculty Cluster Selection Committee, Stony Brook University
2013–present	Undergraduate Curriculum Committee, Computer Science Department, Stony Brook University
2013–present	Member, Provost's Massive Open Online Courses (MOOCS) Task Force, Stony Brook University

Docker EX1003
Page 269 of 310

2012–present	Steering Committee member, The Israeli ACM Experimental Systems Conference (ACM SYSTOR)
2011–2015	Co-Chair, IT Operations committee, Computer Science Department, Stony Brook University
2011–present	Member, University Senate Information Technology Committee, Stony Brook University
2011–present	Member, President's Information Technology Steering Committee on Email and Collaboration, Stony Brook University
2006–2011	Director of IT Operations, Computer Science Department, Stony Brook University
2010–present	Advanced Energy Research and Technology Center (AERTC), Center Director Search Committee member, Stony Brook University
2012	Member, Grant Writer Recruiting committee, Computer Science Department, Stony Brook University
2011	Staff Recruiting committee chair (Windows Server administrator), Computer Science Department, Stony Brook University
2010	Staff Recruiting committee chair (Unix/Networking administrator), Computer Science Department, Stony Brook University
2010	Departmental 5-year Strategic Planning Committee member, Computer Science Department, Stony Brook University
2010	Faculty Recruiting Committee, Computer Science Department, Stony Brook University
2010	Staff Recruiting committee member (Network Administrator), CEWIT Center, Stony Brook University
2010	President's Teaching Excellence Award Selection Committee, Stony Brook University
2009	Staff Recruiting committee chair (Windows client administrator), Computer Science Department, Stony Brook University
2009	Staff Recruiting committee chair (CVC manager), Computer Science Department, Stony Brook University
2009–present	Advanced Energy Research and Technology Center (AERTC), scientific advisor board member, Stony Brook University
2009	President's Teaching Excellence Award Selection Committee, Stony Brook University
2007–2008	Staff Recruiting committee chair (webmaster), Computer Science Department, Stony Brook University
2008–present	New CS building committee, Computer Science Department, Stony Brook University
2007	Staff Recruiting committee chair (Unix administrator), Computer Science Department, Stony Brook University
2007–2010	Co-Chair, IT Operations committee, Computer Science Department, Stony Brook University

2006–present	Center of Excellence in Wireless and Information Technologies (CEWIT) faculty advisory board member, Computer Science Department, Stony Brook University
2006–2007	Chair, Resources and Facilities sub-committee, part of an 8-year departmental review committee. Computer Science Department, Stony Brook University.
2003–present	Visibility (Ranking) committee, Computer Science Department, Stony Brook University
2003–present	Graduate Committee, Computer Science Department, Stony Brook University
2003–present	Permanent Graduate Quals committee, Computer Science Department, Stony Brook University
2003–present	Center of Excellence in Wireless and Information Technologies (CEWIT) committee, Computer Science Department, Stony Brook University
2002–present	Web Site Design committee, Computer Science Department, Stony Brook University
2002–present	Operations Committee, Computer Science Department, Stony Brook University
2001–2006	Faculty Recruiting Committee, Computer Science Department, Stony Brook University
2002–2006	Graduate Recruiting Poster Committee, Computer Science Department, Stony Brook University
2005	SFS Program Director Recruiting committee, Computer Science Department, Stony Brook University
2004	Staff Recruiting committee, Computer Science Department, Stony Brook University
2004	Undergraduate Curriculum Review ad-hoc committee, Computer Science Department, Stony Brook University
2003	PI Award Interface (PIAI) Review committee, Stony Brook University
2003	Graduate Brochure committee, Computer Science Department, Stony Brook University
Spring 2002	Graduate Student Admissions committee, Computer Science Department, Stony Brook University
2002–2003	Graduate Quals Revisions committee, Computer Science Department, Stony Brook University
Spring 2002	Graduate Student Admissions Committee, Computer Science Department, Stony Brook University
Fall 2001	Undergraduate Curriculum Committee, Computer Science Department, Stony Brook University
Fall 2001	Domestic Graduate Student Recruiting Committee, Computer Science Department, Stony Brook University
Fall 2001	Quality of Life Committee, Computer Science Department, Stony Brook University
2001	Ph.D. and M.S. Recruiting Committee, Computer Science Department, Stony Brook University

1998–2000	Ph.D. Committee, Columbia University
1998–1999	Ph.D. Admissions Committee, Columbia University
1990–1999	Facilities Committee, Columbia University

Ph.D. Dissertation Defense Committee

Dec 2021	*Appearance Aware Visualization of High Dimensional Data*, Jenny Hyunjung Lee, Computer Science Department, Stony Brook University.
Aug 2020	Committee Chair, Muhammad Wajahat, *Cost-Efficient Dynamic Management of Cloud Resources via Supervised Learning*, Computer Science Department, Stony Brook University.
Jul 2019	Committee chair, *Analytical Approaches for Dynamic Scheduling in Cloud Environments*, Seyyed Ahmad Javadi, Computer Science Department, Stony Brook University.
Jan 2019	Adviser, *A Practical Auto-Tuning Framework for Storage Systems*, Zhen Cao, Computer Science Department, Stony Brook University.
Oct 2017	*Optimizing System Software with B^ε Trees*, William Jannen, Computer Science Department, Stony Brook University.
May 2017	Committee Chair, Yifeng Sun, *Protection Mechanisms for Virtual Machines on Virtualized Servers*, Computer Science Department, Stony Brook University.
Apr 2017	Adviser, *Kurma: Efficient and Secure Multi-Cloud Storage Gateways for Network-Attached Storage*, Ming Chen, Computer Science Department, Stony Brook University.
Sep 2014	Committee Chair, *Achieving Regulatory Compliance in Data Management*, Sumit Bajaj, Computer Science Department, Stony Brook University.
May 2014	Adviser, *GreenDM: A Versatile Tiering Hybrid Drive for the Trade-Off Evaluation of Performance, Energy, and Endurance*, Zhichao Li, Computer Science Department, Stony Brook University.
May 2014	Committee Chair, *Efficient Implementation Techniques for Block-Level Cloud Storage Systems* Dilip Simha, Computer Science Department, Stony Brook University.
Dec 2013	Adviser, *Multi-dimensional Workload Analysis and Synthesis for Modern Storage Systems*, Vasily Tarasov, Computer Science Department, Stony Brook University.
May 2012	Committee Chair, *Practical Oblivious Outsourced Data Processing*, Peter Williams, Computer Science Department, Stony Brook University.
Jan 2012	Adviser, *Efficient, Scalable, and Versatile Application and System Transaction Management for Direct Storage Layers*, Richard P. Spillane, Computer Science Department, Stony Brook University.
Aug 2009	*Virtualization Mechanisms for Mobility, Security and System Administration*, Shaya Potter, Computer Science Department, Columbia University.
Aug 2009	Adviser, *Flexible Debugging with Controllable Overhead*, Sean Callanan, Computer Science Department, Stony Brook University.
Sep 2008	Committee chair, *Repairable File and Storage Systems*, Ningning Zhu, Computer Science Department, Stony Brook University.

Aug 2008 Adviser, *Analyzing Root Causes of Latency Distributions*, Avishay Traeger, Computer Science Department, Stony Brook University.

Aug 2008 *Programmable Ethernet Switches and Their Applications*, Srikant Sharma, Computer Science Department, Stony Brook University.

May 2008 Committee chair, *Practical Information Flow Based Techniques to Safeguard Host Integrity"*, Weiqing Sun, Computer Science Department, Stony Brook University.

Feb 2008 Adviser, *End-to-End Abstractions for Application-Aware Storage*, Gopalan Sivathanu (advisee), Computer Science Department, Stony Brook University.

Dec 2006 Adviser, *Versatile, Portable, and Efficient File System Profiling*, Nikolai Joukov (advisee), Computer Science Department, Stony Brook University.

Jul 2006 Committee chair, *Availability, Fairness, and Performance Optimization in Storage Virtualization Systems*, Gang "Jason" Peng, Computer Science Department, Stony Brook University.

Jun 2006 *Implementation Techniques for Scalable, Secure and QoS-Guaranteed Enterprise-Grade Wireless LANs*, Fanglu Guo, Computer Science Department, Stony Brook University.

May 2006 Adviser, *Extending ACID Semantics to the File System via* `ptrace`, Charles P. Wright (advisee), Computer Science Department, Stony Brook University.

Aug 2002 *Authoring, Compression, and Playback of Active Video Content*, Anindya Neogi, Computer Science Department, Stony Brook University.

M.S. Thesis Defense Committee

May 2021 *Transfer Learning Techniques for Sequence Labeling in Network File System Specifications*, Amanpreet Singh, Computer Science Department, Stony Brook University.

Dec 2016 Adviser, *To FUSE or not to FUSE? Analysis and Performance Characterization of the FUSE User-Space File System Framework*, Bharath Kumar Reddy Vangoor, Computer Science Department, Stony Brook University.

May 2015 Adviser, *Finding the Right Balance: Security vs. Performance with Network Storage Systems*, Arun Olappamanna Vasudevan, Computer Science Department, Stony Brook University.

May 2012 Adviser, *A Context Aware Block Layer: The Case for Block Layer Deduplication*, Amar Mudrankit, Computer Science Department, Stony Brook University.

May 2012 Adviser, *From Tuples to Files: a Fast Transactional System Store and File System*, Pradeep Shetty, Computer Science Department, Stony Brook University.

December 2010 *TRACECUT: a mechanism of providing history based pointcuts on top of InterAspect*, Ketan Dixit, Computer Science Department, Stony Brook University.

May 2010 Adviser, *Stopping Data Races Using Redflag*, Abhinav Duggal, Computer Science Department, Stony Brook University.

May 2010 Adviser, *Optimizing Energy and Performance for Server-Class File System Workloads*, Priya Sehgal, Computer Science Department, Stony Brook University.

Docker EX1003
Page 273 of 310

May 2010 — *Model Checking the Kaminsky DNS Cache-Poisoning Attack Using PRISM*, Tushar Sudas Deshpande, Computer Science Department, Stony Brook University.

May 2010 — Committee chair, *LFSM (Log Structured Flash Storage Manager)* Gautham Meruva, Computer Science Department, Stony Brook University.

Apr 2009 — Adviser, *The Visual Development of GCC Plug-ins with GDE* Daniel J. Dean, Computer Science Department, Stony Brook University.

Apr 2009 — Adviser, *Energy and Performance Evaluation of Lossless File Data Compression on Computer Systems* Rachita Kothiyal, Computer Science Department, Stony Brook University.

May 2007 — Adviser, *PLEASE: Policy Language for Easy Administration of SELinux*, David P. Quigley, Computer Science Department, Stony Brook University.

Jun 2006 — *Towards the Minimal TCP Offload Architecture*, Lakshmi Kumar Tiruppukuzhi, Computer Science Department, Stony Brook University.

Jun 2006 — *An Approach to Protect System Integrity from Untrusted Applications*, Gaurav Poothia, Computer Science Department, Stony Brook University.

Dec 2005 — Adviser, *Storage Virtualization with a Stackable File System*, Sunil Satnur, Computer Science Department, Stony Brook University.

Dec 2004 — Adviser, *File System Extensibility and Reliability Using an in-Kernel Databases System*, Aditya Kashyap, Computer Science Department, Stony Brook University.

Aug 2004 — Adviser, *Versatile File System Tracing with Tracefs*, Akshat Aranya, Computer Science Department, Stony Brook University.

Mar 2004 — *System and Network Management Using State Machines*, Mohan-Krishna Channa-Reddy, Computer Science Department, Stony Brook University.

Dec 2003 — Adviser, *Versionfs: A Versatile and User-Oriented Versioning File System*, Kiran-Kumar Muniswamy-Reddy (advisee), Computer Science Department, Stony Brook University.

Aug 2003 — *WattProbe: Software-Based Empirical Extraction of Hardware Energy Models*, Manish Prasad, Computer Science Department, Stony Brook University.

Jun 2003 — Adviser, *A System for Improving Application Performance Through System Call Composition*, Amit Purohit, Computer Science Department, Stony Brook University.

Ph.D. Thesis-Proposal (Prelim) Committee

Jun 2022 — Adviser, *Using Machine Learning to Improve Operating Systems' I/O Subsystems*, Ibrahim "Umit" Akgun, Computer Science Department, Stony Brook University.

Apr 2022 — *Exploring Large Parameter Spaces with Machine Learning and Data Visualization*, Anjul Tyagi, Computer Science Department, Stony Brook University.

Apr 2019 — *Cost-Efficient Dynamic Management of Cloud Resources through Supervised Learning*, Muhammad Wajahat, Computer Science Department, Stony Brook University.

Dec 2018 — *Predicting the Future of Archival Storage Using Simulation*, James Byron, Computer Science Department, UC Santa Cruz.

Nov 2018	Committee chair, *Analytical Approaches for Dynamic Scheduling in Cloud Environments*, Seyyed Ahmad Javadi, Computer Science Department, Stony Brook University.
Jun 2018	Committee chair, *Appearance Aware Visualization of High Dimensional Data*, Jenny Hyunjung Lee, Computer Science Department, Stony Brook University.
Apr 2018	Adviser, *A Practical, Real-Time Auto-Tuning Framework for Storage Systems*, Zhen Cao, Computer Science Department, Stony Brook University.
Apr 2017	*Optimizing System Software with B^ε Trees*, William Jannen, Computer Science Department, Stony Brook University.
Nov 2015	Adviser, *KURMA: Geo-Distributed Secure Middlewares for Cloud-Backed Network Attached Storage*, Ming Chen, Computer Science Department, Stony Brook University.
Dec 2014	*Economic Modeling of Long-Term Digital Storage*, Preeti Gupta, Computer Science Department, the University of California at Santa Cruz.
Oct 2013	*Achieving Regulatory Compliance in Data Management*, Sumit Bajaj, Computer Science Department, Stony Brook University.
Aug 2013	Committee chair, *Implementation Techniques for Software-Defined Distributed Storage Systems*, Dilip Simha, Computer Science Department, Stony Brook University.
May 2013	Adviser, *System-Aware Resource Scheduling for Performance, Energy, and Reliability in Tiered Storage Systems*, Zhichao Li, Computer Science Department, Stony Brook University.
Apr 2013	Adviser, *Multi-dimensional Workload Analysis and Synthesis for Modern Storage Systems*, Vasily Tarasov, Computer Science Department, Stony Brook University.
Jan 2012	Adviser, *Runtime Verification of Kernel-Level Concurrency Using Compiler-Based Instrumentation*, Justin Seyster, Computer Science Department, Stony Brook University.
Aug 2011	Adviser, *Efficient, Scalable, and Versatile Application and System Transaction Management for Direct Storage Layers* Richard P. Spillane, Computer Science Department, Stony Brook University.
May 2011	*Practical Oblivious Outsourced Data Processing* Peter Williams, Computer Science Department, Stony Brook University.
Dec 2008	*Organizing, Indexing, and Searching Large-Scale File Systems* Andrew Leung, Computer Science Department, the University of California at Santa Cruz.
Mar 2008	Adviser, *Remote Debugging with Controllable Overhead* Sean Callanan, Computer Science Department, Stony Brook University.
Feb 2008	Adviser, *Analyzing Root Causes of Latency Distributions*, Avishay Traeger, Computer Science Department, Stony Brook University.
Oct 2007	*Practical Information Flow Based Techniques to Safeguard Host Integrity"*, Weiqing Sun, Computer Science Department, Stony Brook University.
Aug 2007	Committee chair, *OS-level Virtualization and Its Applications*, Yang Yu, Computer Science Department, Stony Brook University.
May 2007	Adviser, *End-to-End Abstractions for Application-Aware Storage*, Gopalan Sivathanu, Computer Science Department, Stony Brook University.

Docker EX1003
Page 275 of 310

Jun 2006 — *Building Enterprise-Grade Wireless LAN with High Mobility, Capacity and Security*, Fanglu Guo, Computer Science Department, Stony Brook University.

Apr 2006 — Adviser, *Versatile, Portable, and Efficient File System Profiling*, Nikolai Joukov, Computer Science Department, Stony Brook University.

Jan 2006 — Adviser, *Extending ACID Semantics to the File System*, Charles P. Wright, Computer Science Department, Stony Brook University.

Aug 2005 — Committee chair, *Repairable File and Storage Systems*, Ningning Zhu, Computer Science Department, Stony Brook University.

Dec 2004 — Committee chair, *Programmable Ethernet switch networks and their applications*, Srikant Sharma, Computer Science Department, Stony Brook University.

Sep 2004 — Committee chair, *Availability Support and Performance Optimization in Stonehenge*, Gang "Jason" Peng, Computer Science Department, Stony Brook University.

Mar 2002 — *Stonehenge: A High Performance Virtualized Storage Cluster With QoS Guarantees*, Lan Huang, Computer Science Department, Stony Brook University.

Oct 2001 — *A System for Collaborative Web Resource Categorization and Ranking*, Maxim Lifantsev.

May 2001 — *Authoring, Compression, and Playback of Active Video Content*, Anindya Neogi, Computer Science Department, Stony Brook University.

Ph.D. Research Proficiency Exam (RPE) Committee

Apr 2022 — Committee chair, *Optimizing Near-Data Processing for Spark*, Sri Pramodh Rachuri, Computer Science Department, Stony Brook University.

Jan 2022 — Adviser, *Model-Checking Support for File System Development*, Yifei Liu, Computer Science Department, Stony Brook University.

Aug 2020 — *An Experimental Evaluation of Cache-Adaptive Algorithms*, Abiyaz Chowdhury, Computer Science Department, Stony Brook University.

May 2019 — Adviser, *Re-Animator: Versatile High-Fidelity System-Call Tracing and Replaying* Ibrahim "Umit" Akgun, Computer Science Department, Stony Brook University.

Aug 2018 — Committee chair, *Reducing tail latency in web applications by addressing variability in service times*, Amoghavarsha Suresh, Computer Science Department, Stony Brook University.

Apr 2018 — Committee chair, *Realizing an Elastic Memcached via Cached Data Migration*, Ubaid Ullah Hafeez, Computer Science Department, Stony Brook University.

Sep 2016 — Dynamic Interference-Aware Load Balancing, Seyyed Ahmad Javadi, Computer Science Department, Stony Brook University.

May 2016 — Committee chair, *Leveraging Machine Learning for Black-Box Autoscaling*, Muhammad Wajahat, Computer Science Department, Stony Brook University.

Apr 2016 — *Spatial Big Data Management and Analytics on Cloud environments and modern computing infrastructures* Hoang Vo, Computer Science Department, Stony Brook University.

Docker EX1003
Page 276 of 310

Jan 2016	Adviser, *Parametric Optimization of Storage Systems* Zhen Cao, Computer Science Department, Stony Brook University.
Dec 2015	Adviser, *Design and Implementation of an Open-Source Deduplication Platform for Research* Sonam Mandal, Computer Science Department, Stony Brook University.
Sep 2012	*The Design and Application of Content Addressable Storage Systems*, William Jannen Computer Science Department, Stony Brook University.
Aug 2012	Committee chair, *Interaction for Visualization of Multidimensional Data*, Puripant Ruchikachorn, Computer Science Department, Stony Brook University.
Nov 2011	Committee chair, *Trusted Hardware in Secure Data Management*, Sumit Bajaj, Computer Science Department, Stony Brook University.
Sep 2011	Committee chair, *Isolated and Efficient Execution of Unmodified Network Device Drivers*, Yifeng Sun, Computer Science Department, Stony Brook University.
Sep 2011	Committee chair, *The Art of Data Deduplication*, DilipSimha N M, Computer Science Department, Stony Brook University.
Sep 2011	Adviser, *Power and Performance in Compression Systems: A Control Theoretical Approach with Evaluation*, Zhichao Li, Computer Science Department, Stony Brook University.
Sep 2011	Committee chair, *Isolated and Efficient Execution of Unmodified Network Device Drivers*, Sun Yifeng, Computer Science Department, Stony Brook University.
Sep 2011	*Defensive Techniques for Untrustworthy Operating Systems*, Committee chair, Rui Qiao, Computer Science Department, Stony Brook University.
Aug 2011	*Computational Modeling and Analysis of Cardiac Arrhythmia*, Abhishek Murthy, Computer Science Department, Stony Brook University.
Feb 2010	Adviser, *An End-to-End Performance Evaluation of NFSv4 Implementations*, Vasily Tarasov, Computer Science Department, Stony Brook University.
Aug 2009	*Attacks and Defenses: Unix File-System race conditions*, Xiang Cai, Computer Science Department, Stony Brook University.
Mar 2008	Adviser, *Techniques for Visualizing Software Execution*, Justin Seyster, Computer Science Department, Stony Brook University.
May 2006	Committee chair, *Challenges of Long-Term Digital Archiving*, Maohua Lu, Computer Science Department, Stony Brook University.
Feb 2006	*A Survey of Program Transformation Languages and Systems*, Michael Gorbovitski, Computer Science Department, Stony Brook University.
Jun 2005	*Peer to Peer File Download and Streaming*, Gang Wu, Computer Science Department, Stony Brook University.
May 2005	Adviser, *Debugging and Optimizing Systems With Runtime Instrumentation*, Sean Callanan, Computer Science Department, Stony Brook University.
May 2005	Adviser, *File System Benchmarking: Fallacies and Pitfalls*, Avishay Traeger, Computer Science Department, Stony Brook University.

May 2005 Adviser, *On the Role of Static Analysis in Operating System Checking and Runtime Verification*, Abhishek Rai, Computer Science Department, Stony Brook University.

May 2005 Adviser, *Ensuring Data Integrity in Storage: Techniques and Applications*, Gopalan Sivathanu, Computer Science Department, Stony Brook University.

Feb 2005 *A Survey on Virtualization Techniques*, Susanta Nanda, Computer Science Department, Stony Brook University.

Sep 2004 *Enterprise Digital Rights Management: Solutions against Information Theft by Insiders*, Yang Yu, Computer Science Department, Stony Brook University.

May 2004 Adviser, *Operating System Support for Extensible Secure File Systems*, Charles P. Wright, Computer Science Department, Stony Brook University.

Feb 2004 *Traffic Analysis: From Stateful Firewall to Network Intrusion Detection System*, Fanglu Guo, Computer Science Department, Stony Brook University.

Feb 2004 *Damage Recovery from Malicious Code and Operator Errors*, Weiqing Sun, Computer Science Department, Stony Brook University.

Sep 2003 Adviser, *Internet Worms as Internet-Wide Threat*, Nikolai Joukov, Computer Science Department, Stony Brook University.

Mar 2003 *Cache Oblivious Algorithms: Theory and Practice*, Piyush Kumar, Computer Science Department, Stony Brook University.

Jan 2003 Committee chair, *CDN: Content Distribution Network*, Gang "Jason" Peng, Computer Science Department, Stony Brook University.

Aug 2001 *Pattern Based Intrusion Detection Systems*, Prem Uppuluri, Computer Science Department, Stony Brook University.

Other One-Time Committee Service

2019 Office of Vice-President for Research, Seed Grant review committee Bi-annual Awards committee, Stony Brook University.

Aug 2012 Bi-annual Awards committee, Computer Science Department, Stony Brook University.

Nov 2011 Member, Steering Committee on High-Performance Computing, Stony Brook University and Brookhaven National Labs.

Oct 2010 Annual Merit Raise committee, Computer Science Department, Stony Brook University.

Aug 2010 Bi-annual Awards committee, Computer Science Department, Stony Brook University.

Oct 2009 Annual Merit Raise committee, Computer Science Department, Stony Brook University.

Sep 2008 Annual Merit Raise committee, Computer Science Department, Stony Brook University.

Sep 2007 Annual Merit Raise committee, Computer Science Department, Stony Brook University.

Jan 2007 Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.

Docker EX1003
Page 278 of 310

Jul 2006	Bi-annual Awards committee, Computer Science Department, Stony Brook University.
Apr 2006	Judge, *Compilers* session. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
Sep 2005	Annual Merit Raise committee, Computer Science Department, Stony Brook University.
Aug 2005	Ad-hoc Steering Committee on Strategic Planning, Computer Science Department, Stony Brook University.
May 2004	Judge, *Security* session. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
May 2004	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.
Jan 2004	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.
May 2003	Judge, *Security* session. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
Oct 2003	Ph.D. qualifying exam chair (Hardware exam), Computer Science Department, Stony Brook University.
Oct 2003	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.
May 2003	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.
May 2002	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.
May 2001	Examiner for CS Ph.D. Qualifying Exams, Computer Science Department, Stony Brook University.

Other Departmental Service

2008–present	Course Coordinator, Systems Administration, (ISE-311)
2003–present	Course Coordinator, Introduction to Programming in C (CSE-130)
2002–present	Course Coordinator, Advanced Systems Programming in Unix/C (CSE-376)
2001–present	Course Coordinator, Operating Systems (CSE-506)
2003–2005	Course Coordinator, Intermediate Programming in C and C++ (CSE-230)

Community Work

Nov 2008	"How Computers Work," guest speaker at Mount Elementary 4th grade (teacher: Jo-Ann Hudecek), Stony Brook Three Village School District, Stony Brook, NY.
2003	Stony Brook Child Care Services Inc. (SBCCSI), fund-raising committee member
2002–present	Faculty sponsor of the Stony Brook Linux Users Group (SBLUG)
1997–present	Maintainer of FiST, a stackable file system template system

1992–present	Maintainer of Amd, the Berkeley Automounter (am-utils)

Shepherding

2016–2017	The 15th USENIX Conference on File and Storage Technologies (FAST '17) (1 paper)
2011	The 2011 Usenix Annual Technical Conference (Usenix ATC 2011) (1 paper)
2009	The 2009 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2009) (2 papers)
2009	Fifth USENIX Conference on File and Storage Technologies (FAST '09) (1 paper)
2007	USENIX FAST Annual Conference (2 papers)
2003	USENIX FreeNIX Annual Conference (2 papers)
2002	USENIX FreeNIX Annual Conference (4 papers)
2001	First USENIX Conference on File and Storage Technologies (FAST '02) (1 paper)

Software Released or Maintained

2010–present	Wrapfs Stackable File System for Linux
2007–present	GCC Plugin Infrastructure.
2005–present	Auto-Pilot: A Suite of Tools for Conducting Benchmarks, `www.filesystems.org/project-autopilot.html`
2004–present	Unionfs: A Stackable Unification File System, `https://unionfs.filesystems.org`. (As of January 2007, Unionfs has been a part of Andrew Morton's Linux Kernel source tree. Andrew is the person in charge of supplying new patches to Linus Torvalds for new kernel releases, therefore Unionfs is on its way to becoming a part of the Linux kernel.)
1997–present	FiST: Stackable File System Language and Templates, `www.filesystems.org/`
1992–present	Am-Utils: The Berkeley Automounter Suite of Utilities, `www.am-utils.org`
2004–2005	I^3FS: In-Kernel Integrity Checker and Intrusion Detection File System, `www.fsl.cs.sunysb.edu/project-i3fs.html`
2003–2004	KBDB: In-Kernel Berkeley DB Databases NFScrack: NFS File Handle Security, KBDBFS: File System Extensibility and Reliability Using an in-Kernel Database, `www.fsl.cs.sunysb.edu/project-kbdb.html`
2002	ENF: Enhancing NFS Cross-Administrative Domain Access, `ftp://ftp.fsl.cs.sunysb.edu/pub/enf/`
1999	PGmake: Parallel GNU Make, `www.fsl.cs.sunysb.edu/project-pgmake.html`

TALKS

Invited Talks

May 2017	*Oh, what a tangled web we weave: practice and deception in system analysis*, Performance Evaluation and Analysis Meetup, IBM Research–Haifa, Haifa, Israel.
May 2017	*vNFS: Maximizing NFS Performance with Compounds and Vectorized I/O*, ACM SYSTOR 2017 (highlights paper track), Haifa, Israel.

Feb 2017	*Oh, what a tangled web we weave: practice and deception in system analysis*, Tintri, Inc., Mountain View, CA.
Dec 2016	*Oh, what a tangled web we weave: practice and deception in system analysis*, Harvey Mudd College, Claremont, CA.
Jun 2016	*Oh, what a tangled web we weave: practice and deception in system analysis* Keynote address at inaugural workshop: The 1st International Workshop on System Analytics and Characterization (SAC'16, co-located with ACM SIGMETRICS), Antibes Juan-Les-Pins, France.
Feb 2016	*Parametric Optimization of Storage Systems*, VMware Inc., Palo Alto, CA.
Feb 2006	*File System and Storage Benchmarking: On Formalizing Filebench Workloads*, a Birds-of-Feather (BoF) talk presented at the 2016 Usenix FAST conference, Santa Clara, CA.
Jun 2012	*Storage Security Overview*, CA/CEWIT center inauguration workshop, Tel Aviv University, Tel Aviv, Israel.
May 2012	*Synthesizing Realistic Workload Models and Dedup Datasets*, IBM Haifa Research Labs, Tel Aviv, Israel.
May 2012	*Optimizing Performance and Energy Use of Large Data Systems*, NSF Industry/University Center (I/UCRC) for Dynamic Data Analytics (CDDA) 2012 workshop, Stony Brook, NY.
Feb 2012	*Scaling OS Storage Stack Performance Using NVRAM Technologies*, Fusion-io, San Jose, CA
Nov 2011	*Scaling and Understanding Large Data I/O Performance*, NSF Industry/University Center (I/UCRC) for Dynamic Data Analytics (CDDA) 2011 workshop, Rutgers, NJ.
Nov 2011	*Scaling I/O Performance for Large Data Sets*, 2011 CEWIT Center Conference, Hauppauge, NY.
Nov 2011	*Scaling OS Storage Stack Performance Using NVRAM Technologies*, Operating System Support for Next Generation Large Scale NVRAM (NVRAMOS'11), Jeju Island, South Korea.
Aug 2011	*Optimizing Server Performance and Energy*, High-End Computing Filesystems and I/O Workshop (HEC FSIO 2011), Arlington, VA.
May 2011	*On the Energy Consumption and Performance of Systems Software*, The 4th Israeli Experimental Systems Conference (ACM SYSTOR '11), Haifa, Israel.
Feb 2011	*FSL Storage Research Overview*, Riverbed, Inc, Menlo Park, CA.
Nov 2010	*Server-Class Energy and Performance Evaluations, or, Mommy, Daddy, where did my Joules go?*, An SBCS Faculty Symposium series talk, sponsored by the Stony Brook Computer Society's ACM Chapter. Stony Brook University, Computer Science Department, Stony Brook, NY.
May 2010	*Server-Class Energy and Performance Evaluations*, The Third Israeli Experimental Systems Conference (ACM SYSTOR '10), Haifa, Israel.

Docker EX1003
Page 281 of 310

Feb 2010	*On the Science of Power Management: Encouraging Sustainability R&D*, First USENIX Workshop on Sustainable Information Technology (SustainIT'10), San Jose, CA.
Feb 2010	*FSL Research Overview*, Data Domain, Inc, Santa Clara, CA.
Nov 2009	Priya Sehgal and Erez Zadok, *Improving Energy & Performance For Server Workloads*, The third Advanced Energy Research & Technology Center (AERTC) conference, Hauppauge, NY.
July 2009	*Energy and Performance Characterization of Storage Stacks*, IBM Watson Research Labs, Hawthorne, NY.
May 2009	*DHIS: Discriminating Hierarchical Storage*, The Israeli Experimental Systems Conference (ACM SYSTOR '09), Haifa, Israel.
May 2009	*Energy and Performance Evaluation of Lossless File Data Compression on Server Systems*, The Israeli Experimental Systems Conference (ACM SYSTOR '09), Haifa, Israel.
February 2009	*How to Cheat in Benchmarking*, a Birds-of-Feather (BoF) talk presented at the 2009 Usenix FAST conference, San Francisco, CA.
Nov 2008	*Mommy, Daddy, Why is My New Laptop So Dog Slow?!*, Parents' Day lecture, Stony Brook University, Stony Brook, NY.
May 2007	*Why is my new laptop so dog slow?!*, An SBCS Faculty Symposium series talk, sponsored by the Stony Brook Computer Society's ACM Chapter. Stony Brook University, Computer Science Department, Stony Brook, NY.
July 2006	*Securing Applications using Operating System Transactions*, Center for Information Protection, NSF I/UCRC organizational workshop, Stony Brook, NY.
June 2006	*Adding security to file systems, one layer at a time*, IBM's Storage Systems Strategy meeting, Mountain View, CA.
June 2006	*File System Benchmarking: Fallacies, Pitfalls, and Beyond*, VMware, Inc., Palo Alto, CA.
December 2005	*File System Benchmarking and Tools: Past, Present, and Future*, a Birds-of-Feather (BoF) talk presented at the 2006 Usenix FAST conference, San Francisco, CA.
December 2005	*File System Benchmarking: Fallacies, Pitfalls, and Beyond*, Penn State University, Computer Science Department, State College, PA.
October 2005	*File System Benchmarking: Fallacies, Pitfalls, and Beyond*, IBM Watson Research Labs, Hawthorne, NY.
October 2005	*File System Benchmarking: Fallacies, Pitfalls, and Beyond*, CSE-600 talk, Stony Brook University, Computer Science Department, Stony Brook, NY.
September 2005	*File System Benchmarking: Fallacies, Pitfalls, and Beyond*, UC Santa Cruz, Computer Science Department, Santa Cruz, California.
June 2005	*Adding security to file systems, one layer at a time*, Second Security Summit, Storage Networking Industry Association (SNIA), Pittsburgh, Pennsylvania
May 2005	*File System Security*, 35th Anniversary Celebration, Stony Brook University, Computer Science Department (CS@35), Stony Brook, New York

Docker EX1003
Page 282 of 310

April 2005	*Efficient and Safe Execution of User-Level Code in the Kernel*, the 2005 NSF Next Generation Software Workshop, in conjunction with the 2005 International Parallel and Distributed Processing Symposium (IPDPS), Denver, Colorado
October 2004	*File System and OS Security*, talk given to Korean delegation led by Dr. Ho, Stony Brook University, Stony Brook, New York
October 2004	*File System and OS Security*, Computer Associates, Islandia, New York
June 2003	*ASK: Active Spam Killer*, Annual Usenix Conference, FREENIX Track, San Antonio, Texas
February 2003	*File System Security*, New York State Cyber-Security Symposium, Utica, New York
April 2000	*FiST: A System for File System Code Generation*, Computer Science Department, University of Chicago
March 2000	*FiST: A System for File System Code Generation*, Computer Science Department, University of Rochester
March 2000	*FiST: A System for File System Code Generation*, Computer Science Department, Stony Brook University
March 2000	*FiST: A System for File System Code Generation*, Computer Science Department, University of California and Santa Barbara
March 2000	*FiST: A System for File System Code Generation*, Computer Science Department, Penn State University
February 2000	*FiST: A System for File System Code Generation*, Computer Science Department, Bell Labs, New Jersey

Interviews

November 4, 2005	LIA magazine, with Ellen Sterling; what small businesses can do to protect themselves.
June 29, 2001	Dr. Dobb's Technetcast, with Phillippe Lourier; my NFS and Automounters book, status of Linux file system development.

Other Presentations

Mar 2012	Erez Zadok, Steering Committee Town Hall meeting, presenting Google Apps for Education, Stony Brook University.
Feb 2012	Daniel C. Rosenthal, University of California, Santa Cruz; David S. H. Rosenthal, Stanford University Libraries; Ethan L. Miller and Ian F. Adams, University of California, Santa Cruz; Mark W. Storer, NetApp; Erez Zadok, Stony Brook University. *Toward an Economic Model of Long-Term Storage* WIP presented at the *Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA.
Feb 2012	Zhichao Li, Stony Brook University; Kevin M. Greenan and Andrew W. Leung, EMC Corporation; Erez Zadok, Stony Brook University. *Power Consumption in Enterprise-Scale Backup Storage Systems* Poster presented at the *Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA.
Feb 2012	V. Tarasov and S. Kumar, Stony Brook University; J. Ma, Harvey Mudd College; D. Hildebrand and A. Povzner, IBM Almaden Research; G. Kuenning, Harvey Mudd College; E. Zadok, Stony Brook University *Extracting Flexible, Replayable Models from*

Docker EX1003
Page 283 of 310

Large Block Traces Poster presented at the *Tenth USENIX Conference on File and Storage Technologies (FAST '12)*, San Jose, CA.

Nov 2011 — Zhichao Li, R. Grosu, S. A. Smolka, S. D. Stoller, and E. Zadok, Stony Brook University. *Power and Performance in Compression Systems: A Control Theoretical Approach with Evaluation* Poster presented at the *Eighth CEWIT conference (CEWIT '11)*, Hauppauge, NY.

Oct 2011 — Zhichao Li, R. Grosu, S. A. Smolka, S. D. Stoller, and E. Zadok, Stony Brook University. *Power and Performance in Compression Systems: A Control Theoretical Approach with Evaluation* Poster presented at the *Fifth AERTC conference (AERTC '11)*, Buffalo, NY.

May 2011 — Vasily Tarasov, Koundinya Santhosh Kumar, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College. *T2M: Converting I/O Traces to Workload Models*, Poster presented at the *13th USENIX Workshop on Hot Optics in Operating Systems (HotOS '11)*, Napa, CA.

Feb 2011 — Vasily Tarasov, Koundinya Santhosh Kumar, and Erez Zadok, Stony Brook University; Geoff Kuenning, Harvey Mudd College. *T2M: Converting I/O Traces to Workload Models*, WIP and Poster presented at the *Ninth USENIX Conference on File and Storage Technologies (FAST '11)*, San Jose, CA.

Feb 2011 — Erez Zadok, Stony Brook University and Geoff Kuenning, Harvey Mudd College. *Benchmarking and Tracing: New Horizons*, Birds-of-a-feather (BOF) presented at the *Ninth USENIX Conference on File and Storage Technologies (FAST '11)*, San Jose, CA.

Oct 2011 — Justin Seyster, Abhinav Duggal, Prabakar Radhakrishnan, Scott D. Stoller, and Erez Zadok, Stony Brook University. *Redflag: Detailed Runtime Analysis of Kernel-Level Concurrency*, Poster presented at the *9th USENIX Symposium on Operating Systems Design and Implementations (OSDI '10)*, Vancouver, BC, Canada.

Feb 2010 — Vasily Tarasov, Sujay Godbole, and Erez Zadok. *NSFv4 Implementations: Who Performs Better, When, and Why*, poster presented at the *Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, San Jose, CA.

Feb 2010 — Rick Spillane and Erez Zadok. *Down with the VFS and His Insidious Caches!*, WIP presented at the *Eighth USENIX Conference on File and Storage Technologies (FAST '10)*, San Jose, CA.

Jan 2010 — Erez Zadok and Radu Sion. *Energy and Cost Analysis of Cloud Computing*, NSF Industry/University Center (I/UCRC) for Dynamic Data Analytics (CDDA) planning workshop, New York, NY.

Oct 2009 — Priya Sehgal, Vasily Tarasov, and Erez Zadok. *Server Energy and Performance Evaluations*, poster presented at the *2009 CEWIT Center Conference*, Islandia, NY.

Dec 2008 — Rick Spillane, Chaitanya Yalamanchili, Sachin Gaikwad, Manjunath Chinni, and Erez Zadok. *Honor: A Serializing On-Disk Writeback Buffer*. WIP presented at the *Eighth Symposium on Operating Systems Design and Implementation (OSDI '08)*, San Diego, CA.

Mar 2007 — D. Dean, S. Callanan, and E. Zadok. *Developing GCC Transformations Faster with*

Docker EX1003
Page 284 of 310

Gimple Viz Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

Mar 2007 — J. Seyster, S. Callanan, R. Grosu, S. Smolka, and E. Zadok. *Memcov A High Performance Memory Profiling Tool*. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

Mar 2007 — R. Spillane and E. Zadok. *A Lot for a Little: Transactional File Systems*. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

Apr 2006 — R. Spillane, C. P. Wright, and E. Zadok, *Goanna* (Best Paper Award), Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

Apr 2006 — N. Joukov, H. Papaxenopoulos, and E. Zadok, *Secure Deletion Myths, Issues, and Solutions*. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

Sep 2005 — E. Zadok, *A Layered Approach to Securing File Systems*, poster presented at the NSF CyberTrust PI meeting, Newport Beach, CA.

Oct 2005 — N Joukov, R. Iyer, A. Traeger, C. P. Wright, and E. Zadok. Versatile, Portable, and Efficient OS Profiling via Latency Analysis. Poster presented at the *20th ACM Symposium on Operating Systems Principles (SOSP'05)*, Brighton, UK.

Jul 2005 — D. Quigley, C. P. Wright, and E. Zadok. *UnionFS: Namespace Management*. Cyber Corps Summer Symposium 2005, Syracuse, New York.

Apr 2005 — D. Quigley, C. P. Wright, and E. Zadok. *UnionFS*. URECA (undergraduate research) presentation at Stony Brook.

Aug 2004 — Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An On-Access Anti-Virus File System. Poster presented at the *Workshop on Information Assurance Education* (WIAED 2004), Stony Brook University

Aug 2004 — C. P. Wright, M. C. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. Poster presented at the *Workshop on Information Assurance Education* (WIAED 2004), Stony Brook University

May 2004 — P. Gupta, H. Krishnan, C. P. Wright, and E. Zadok. Versatility and Unix Semantics in a Fan-Out Unification File System. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

May 2004 — A. Kashyap, J. Dave, H. Krishnan, C. P. Wright, M. Nayyer Zubair, and E. Zadok. Using Berkeley DB in the Linux Kernel. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

May 2004 — Y. Miretskiy, A. Das, C. P. Wright, and E. Zadok. Avfs: An On-Access Anti-Virus File System (Best Paper Award). Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.

May 2004 — A. Aranya, C. P. Wright, and E. Zadok Tracefs: A File System to Trace Them All. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University

Apr 2004 — C. P. Wright and E. Zadok. A Stackable Smorgasbord. Poster presented at the *Third USENIX Conference on File and Storage Technologies (FAST '04)*, San Francisco, CA.

Docker EX1003
Page 285 of 310

Apr 2004	C. P. Wright and E. Zadok. *Avfs: An On-Access Anti-Virus File System* WIP presented at the *Third USENIX Conference on File and Storage Technologies (FAST '04)*, San Francisco, CA.
Apr 2004	K. Muniswamy-Reddy, and E. Zadok. Elastic Quotas. Poster presented at the *12th NASA Goddard, 21st IEEE Conference on Mass Storage Systems and Technologies (MSST 2004)*, College Park, MD.
Nov 2003	C. P. Wright and E. Zadok. Secure and Mobile File Systems. Poster presented at the *CEWIT 2003* conference, Woodbury, NY.
June 2003	P. Gupta, H. Krishnan, K. Muniswamy-Reddy, C. P. Wright, and E. Zadok. Unionfs: A Flexible, and Portable Unioning File System. WIP presented at the *Annual USENIX Technical Conference*, San Antonio, TX.
June 2003	A. Himmer, K. Muniswamy-Reddy, C. P. Wright, and E. Zadok. Versionfs: A Transparent and Portable Versioning File System. WIP presented at the *Annual USENIX Technical Conference*, San Antonio, TX.
May 2003	A. Himmer and E. Zadok. Versionfs: A Transparent and Portable Versioning File System. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
May 2003	E. Zadok, J. Osborn, A. Shater, C. P. Wright, K. Muniswamy-Reddy, and J. Nieh. Elastic Quotas: Stretch Your Disk Space. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
May 2003	C. P. Wright, M. C. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
May 2003	A. Purohit, C. P. Wright, J. Spadavecchia, and E. Zadok. Cosy: Develop in User-Land, Run in Kernel-Mode. Graduate Research Conference (GRC), Computer Science Department, Stony Brook University.
April 2003	M. Prasad and E. Zadok. Transparent Page Cache Coherence Support for Linux-based Stackable File Systems. WIP presented at the *Second USENIX Conference on File and Storage Technologies (FAST '03)*, San Francisco, CA.
Fall 2002	E. Zadok. Linux Kernel Internals. Guest lecture in CSE-306.
December 2002	A. Purohit, J. Spadavecchia, C. Wright, and E. Zadok. Improving Application Performance Through System Call Composition WIP presented at the *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA.
December 2002	C. Wright, M. Martino, and E. Zadok. NCryptfs: A Secure and Convenient Cryptographic File System. WIP presented at the *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA.
December 2002	E. Zadok, J. Osborn, C. P. Wright, K. Muniswamy-Reddy. Elastic Quotas. WIP presented at the *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, MA.
June 2002	A. Purohit, J. Spadavecchia, C. Wright, and E. Zadok. Improving Application Performance Through System Call Composition. WIP presented at the *Annual USENIX Technical Conference*, Monterey, CA.

Docker EX1003
Page 286 of 310

June 2002 — O. C. Leonard, J. Nieh, E. Zadok, J. Osborn, A. Shater, and C. Wright. The Design and Implementation of Elastic Quotas: A System for Flexible File System Management. WIP presented at the *Annual USENIX Technical Conference*, Monterey, CA.

January 2002 — J. Spadavecchia and E. Zadok. Enhancing NFS Cross-Administrative Domain Access. WIP presented at the *First USENIX Conference on File and Storage Technologies (FAST '02)*, Monterey, CA.

December 1999 — E. Zadok and J. Nieh. FiST: A Language for Stackable File Systems. Poster presented at the *17th ACM Symposium on Operating Systems Principles (SOSP)*, Kiawah Island, SC.

TEACHING

Instructor

Spring 2022 — Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University

Spring 2022 — Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University

Spring 2022 — Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University

Spring 2022 — Advanced Project in Computer Science I (CSE-524), Computer Science Department, Stony Brook University

Spring 2022 — Graduate Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University

Spring 2022 — Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University

Fall 2021 — Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University

Fall 2021 — Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University

Fall 2021 — Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University

Fall 2021 — M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University

Fall 2021 — Seminar in Concurrency (CSE-643), Computer Science Department, Stony Brook University

Fall 2021 — Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University

Summer 2021 — M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University

Summer 2021 — Internship in Research (CSE-696), Computer Science Department, Stony Brook University

Docker EX1003
Page 287 of 310

Spring 2021	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2021	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Spring 2021	Advanced Project in Computer Science I (CSE-524), Computer Science Department, Stony Brook University
Spring 2021	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2021	Graduate Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Spring 2021	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2020	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Fall 2020	Senior Honors Research Project II (CSE-499), Computer Science Department, Stony Brook University
Fall 2020	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2020	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2020	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2020	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Summer 2020	Senior Honors Research Project I (CSE-495), Computer Science Department, Stony Brook University
Summer 2020	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Spring 2020	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2020	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Spring 2020	Advanced Project in Computer Science I (CSE-524), Computer Science Department, Stony Brook University
Spring 2020	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2020	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University

Fall 2019	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Fall 2019	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2019	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2019	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2019	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2019	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2019	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2019	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Spring 2019	Advanced Project in Computer Science I (CSE-524), Computer Science Department, Stony Brook University
Spring 2019	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2019	Graduate Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Winter 2019	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2018	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Fall 2018	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Fall 2018	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2018	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2018	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2018	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Summer 2018	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University

Summer 2018	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2018	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Spring 2018	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2018	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Spring 2018	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2018	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2018	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2017	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Fall 2017	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2017	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2017	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Summer 2017	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2017	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2017	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2017	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University
Spring 2017	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2017	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2016	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2016	Advanced Project in Computer Science I (CSE-523), Computer Science Department, Stony Brook University

Fall 2016	Advanced Project in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2016	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2016	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2016	Graduate Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2016	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2016	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2016	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2016	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2016	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Spring 2016	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2016	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2016	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2016	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2015	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2015	Senior Honors Research Project II (CSE-496), Computer Science Department, Stony Brook University
Fall 2015	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2015	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2015	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2015	Graduate Teaching Practicum (CSE-698), Computer Science Department, Stony Brook University

Fall 2015	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2015	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2015	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2015	Topics in Computer Science: Storage Systems (CSE-593), Computer Science Department, Stony Brook University
Spring 2015	Senior Honors Research Project I (CSE-495), Computer Science Department, Stony Brook University
Spring 2015	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2015	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2015	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2015	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2015	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2014	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Fall 2014	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2014	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2014	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2014	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2014	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2014	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2014	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2014	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University

Spring 2014	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2014	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2014	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2014	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2013	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Fall 2013	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2013	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2013	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2013	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2013	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2013	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Summer 2013	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2013	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2013	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2013	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2013	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2013	Topics in Computer Science: Storage Systems (CSE-595), Computer Science Department, Stony Brook University
Spring 2013	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2012	*Operating Systems* (CSE-306), Computer Science Department, Stony Brook University
Fall 2012	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University

Fall 2012	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2012	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2012	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2012	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2012	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Summer 2012	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2012	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2012	Special Project in Computer Science (CSE-522), Computer Science Department, Stony Brook University
Spring 2012	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2012	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2012	Topics in Computer Science: Storage Systems (CSE-595), Computer Science Department, Stony Brook University.
Spring 2012	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2012	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2011	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Fall 2011	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2011	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2011	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2011	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2011	Information Systems Internship (ISE-488), Computer Science Department, Stony Brook University

Summer 2011	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2011	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Spring 2011	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2011	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2011	*M.S. Thesis Research* (CSE-599), Computer Science Department, Stony Brook University
Spring 2011	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2010	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2010	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2010	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2010	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2010	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2010	M.S. Internship in Research (CSE-596), Computer Science Department, Stony Brook University
Summer 2010	Internship in Research (CSE-696), Computer Science Department, Stony Brook University
Summer 2010	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2010	Systems Administration (CSE-311), Computer Science Department, Stony Brook University
Spring 2010	Systems Administration (ISE-311), Computer Science Department, Stony Brook University
Spring 2010	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2010	*M.S. Thesis Research* (CSE-599), Computer Science Department, Stony Brook University
Spring 2010	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University

Spring 2010	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2009	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2009	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2009	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2009	*M.S. Thesis Research* (CSE-599), Computer Science Department, Stony Brook University
Fall 2009	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2009	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Spring 2009	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2009	Undergraduate Teaching Practicum (CSE-475), Computer Science Department, Stony Brook University
Spring 2009	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2009	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2009	Proficiency Requirement in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Spring 2009	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2009	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2009	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2008	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2008	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2008	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2008	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University

Fall 2008	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2008	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2008	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2008	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2008	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2008	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2008	Proficiency Requirement in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Spring 2008	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2008	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Spring 2008	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2007	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2007	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2007	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2007	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2007	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2007	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2007	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2007	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2007	Undergraduate Teaching Practicum (CSE-475), Computer Science Department, Stony Brook University

Docker EX1003
Page 297 of 310

Spring 2007	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2007	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2007	Topics in Computer Science: Secure Storage (CSE-590), Computer Science Department, Stony Brook University. Co-taught with Prof. Radu Sion.
Spring 2007	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2007	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2007	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2006	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2006	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2006	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2006	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2006	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2006	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Summer 2007	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2006	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2006	Topics in Computer Science: File Systems (CSE-594), Computer Science Department, Stony Brook University
Spring 2006	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2006	Proficiency Requirement in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Spring 2006	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2006	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University

Fall 2005	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Fall 2005	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2005	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2005	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2005	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2005	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Spring 2005	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Summer 2005	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Summer 2005	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Summer 2005	FT Summer Research (CSE-800), Computer Science Department, Stony Brook University
Spring 2005	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Spring 2005	Advanced Operating Systems (CSE-624), Computer Science Department, Stony Brook University
Spring 2005	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Spring 2005	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Spring 2005	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2005	Proficiency Requirement in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Spring 2005	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2005	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2004	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University

Fall 2004	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2004	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2004	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Fall 2004	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Spring 2004	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2004	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2004	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Spring 2004	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2004	Practicum in Teaching (CSE-698), Computer Science Department, Stony Brook University
Spring 2004	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2003	Advanced Systems Programming in Unix/C (CSE-376), Computer Science Department, Stony Brook University
Fall 2003	Introduction to Software Engineering and Project Plan (CSE-523), Computer Science Department, Stony Brook University
Fall 2003	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2003	Proficiency Requirement in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Fall 2003	Independent Study in Computer Science (CSE-593), Computer Science Department, Stony Brook University
Fall 2003	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Summer 2003	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Summer 2003	Intro to Software Engineering (CSE-523), Computer Science Department, Stony Brook University
Spring 2003	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University

Spring 2003	Senior Honors Research Project II (CSE-496), Computer Science Department, Stony Brook University
Spring 2003	Intro to Software Engineering (CSE-523), Computer Science Department, Stony Brook University
Spring 2003	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Spring 2003	Independent Study in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Spring 2003	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2003	Graduate Teaching Practicum (CSE-698), Computer Science Department, Stony Brook University
Fall 2002	Advanced Systems Programming in Unix/C (CSE-376), new course developed, Computer Science Department, Stony Brook University
Fall 2002	Senior Honors Research Project I (CSE-495), Computer Science Department, Stony Brook University
Fall 2002	Intro to Software Engineering (CSE-523), Computer Science Department, Stony Brook University
Fall 2002	Lab in Computer Science II (CSE-524), Computer Science Department, Stony Brook University
Fall 2002	Independent Study in Computer Science (CSE-587), Computer Science Department, Stony Brook University
Fall 2002	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Summer 2002	Intro to Software Engineering (CSE-523), Computer Science Department, Stony Brook University
Spring 2002	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2002	Undergraduate Teaching Practicum (CSE-475), Computer Science Department, Stony Brook University
Spring 2002	Research in Computer Science (CSE-487), Computer Science Department, Stony Brook University
Spring 2002	Intro to Software Engineering (CSE-523), Computer Science Department, Stony Brook University
Spring 2002	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Spring 2002	Internship in Research (CSE-698), Computer Science Department, Stony Brook University

Spring 2002	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Fall 2001	*Special Topics in Software Engineering: Systems Programming* (CSE/ISE-391), Computer Science Department, Stony Brook University
Fall 2001	M.S. Thesis Research (CSE-599), Computer Science Department, Stony Brook University
Fall 2001	Ph.D. Dissertation Research (CSE-699), Computer Science Department, Stony Brook University
Spring 2001	Graduate Level *Operating Systems* (CSE-506), Computer Science Department, Stony Brook University
Spring 2001	*Advanced File Systems*, seminar, Computer Science Department, Stony Brook University
November 1999	Tutorial, *Using Amd and the Automounter Utilities*, USENIX LISA 13, Seattle, Washington
October 1999	*NFS: Protocols, Programming and Implementation* (guest lecture in Internet Systems Programming), Computer Science Department, Columbia University
1997–1999	*Operating System Vulnerabilities* (guest lecture in Network Security), Computer Science Department, Columbia University
Fall 1995	*Data Structures in C*, Computer Science Department, Columbia University
1986	*Programming in BASIC*, A.A. Computers, Holon, Israel
1986	*Programming in Logo*, A.A. Computers, Holon, Israel

Teaching Assistant

Spring 1995	*Computer Security*, Computer Science Department, Columbia University
Fall 1994	*Operating Systems*, Computer Science Department, Columbia University
Spring 1992	*Object-Oriented Programming and C++*, Computer Science Department, Columbia University
Spring 1989	*Data Structures*, Computer Science Department, Columbia University
Fall 1989	*Software Design Lab,* Computer Science Department, Columbia University
Fall 1988	*Introduction to Programming in Pascal*, Computer Science Department, Columbia University

GRADUATE STUDENTS (PH.D.)

Ph.D. Students Graduated

May 2014–Jan 2019	Zhen Cao Dissertation title: A Practical Auto-Tuning Framework for Storage Systems Current Position: Google, Inc. (Mountain View, CA)
May 2012–Apr 2017	Ming Chen. Dissertation title: Kurma: Efficient and Secure Multi-Cloud Storage Gateways for Network-Attached Storage Current Position: Google, Inc. (New York, NY)

Sep 2009–May 2014	Zhichao Li. Dissertation title: GreenDM: A Versatile Tiering Hybrid Drive for the Trade-Off Evaluation of Performance, Energy, and Endurance Current Position: VMware, Inc. (Palo Alto, CA)
Jan 2008–Nov 2013	Vasily Tarasov (IBM Ph.D. Fellow). Dissertation title: Multi-dimensional Workload Analysis and Synthesis for Modern Storage Systems Current Position: IBM Research–Almaden, Inc. (San Jose, CA)
Jan 2006–Dec 2013	Justin Seyster (domestic) Dissertation title: Runtime Verification of Kernel-Level Concurrency Using Compiler-Based Instrumentation Current Position: Apple Computer, Inc. (Cupertino, CA)
Jan 2008–Feb 2012	Richard Spillane (domestic). Dissertation title: Efficient, Scalable, and Versatile Application and System Transaction Management for Direct Storage Layers. Current Position: Apple Computer, Inc. (Cupertino, CA)
Sep 2007–May 2011	Vinay Pai (domestic). Dissertation title: Incentive Mechanisms for Peer-to-Peer Streaming. Current Position: MuCash (New York, NY)
Sep 2003–Aug 2009	Sean Callanan (domestic). Dissertation title: Flexible Debugging with Controllable Overhead. Current Position: Apple Computer, Inc. (Cupertino, CA)
Sep 2003–Aug 2008	Avishay Traeger (domestic) (IBM Ph.D. Fellow). Dissertation title: Analyzing Root Causes of Latency Distributions. Current Position: Research Staff Member, IBM Haifa Research Labs (Tel-Aviv, Israel)
Sep 2003–Feb 2008	Gopalan Sivathanu. Dissertation title: End-to-End Abstractions for Application-Aware Storage. Current position: Google, Inc. (Mountain View, CA).
May 2003–Dec 2006	Nikolai Joukov. Dissertation title: Versatile, Portable, and Efficient File System Profiling. Current position: Research Staff Member, IBM T. J. Watson Research Center (Hawthorne, NY).
May 2003–May 2006	Charles P. Wright (domestic). Dissertation title: Extending ACID Semantics to the File System via `ptrace`. Current position: Research Staff Member, IBM T. J. Watson Research Center (Hawthorne, NY).

Current Ph.D. Students

Jan 2020–present	Yifei Liu
May 2019–present	Meyer “Alex” Merenstein
May 2018–present	Tyler Estro
Sep 2017–present	Ibrahim “Umit” Akgun

Past Ph.D. Students Advised

Docker EX1003
Page 303 of 310

May 2019–Dec 2021	Wei Su
Nov 2014–Dec 2017	Sun (Jason) Zhen (visiting scholar from China)
Jun 2014–Dec 2016	Sonam Mandal
May 2004–Nov 2004	Anand Kashyap (He was Samir Das student. I advised him on one project and paper that came out of a class project.)
Sep 2001–May 2002	Li Aili (Student came to program interested in systems work, but became interested in graphics and switched areas altogether. Currently advised by Klaus Mueller.)
May 2001–May 2002	Joseph Spadavecchia (domestic) (I advised and funded student for one year. Student dropped out of program to go work and study in Scotland, for personal reasons. We co-authored one paper together.)
May 2002–Aug 2002	Zhenghong "Sam" Yang (Student already dropped from PhD to MS before I funded him under SPIR. Student graduated and now works for IBM.)
Sep 2001–May 2002	Hui Zhang (Student came to program interested in systems work, but became interested in hardware systems. Currently advised by Yuanyuan Yang.)

GRADUATE STUDENTS (M.S.)

Current M.S. Students

May 2022–present	Pradeep Nalluri
Jun 2021–present	Archit Saxena

Past M.S. Students

Jun 2021–May 2022	Manish Adkar
Jun 2021–May 2022	Nasratullah Sultany
Jun 2021–Jun 2022	Pei Liu
Jun 2021–Dec 2021	Rohan Chhabra
Jun 2021–May 2022	Rishabh Srivastava
Jun 2021–May 2022	Shushanth Madhubalan
Jun 2021–May 2022	Tejeshwar Gurram
Jan 2021–Dec 2021	Aakarsh Duvvuru
Sep 2020–Dec 2021	Aadil Shaikh
Jun 2020–May 2021	Dheeraj Ramchandani
Jan 2020–Dec 2020	Abhiraj Smit
Jan 2020–May 2021	Gomathi Ganesan
Jan 2020–Dec 2020	Vigneshwaran Mylsamy
Jan 2020–Dec 2020	Yatna Verma

Aug 2019–May 2020	Aneesh Joshi
Jun 2019–Dec 2019	Sagar Jeevan
Jan 2019–Dec 2019	Akshay Aurora
Jan 2019–Dec 2019	Jatin Sood
Jan 2019–Dec 2019	Krapi Ravindra Shah
Jan 2019–Dec 2019	Mehul Jain
Jan 2019–Dec 2019	Pragesh Jagnani
Jan 2019–Dec 2019	Shobhit Khandelwal
Sep 2018–Dec 2019	Aayush Sureka
Sep 2018–May 2019	Dhivahar Perumal
Sep 2018–Dec 2019	Ritika Nevatia
May 2018–Dec 2019	Nilesh Somani
Jan 2018–Dec 2019	Prafful Agarwal
Jan 2018–Dec 2018	Ankit Aggarwal
Jan 2018–Dec 2018	Dhanashri Patil
Jan 2018–Dec 2018	Manu Mathew
Jan 2018–Dec 2018	Prateek Roy
Jan 2018–Dec 2018	Rahul Rane
Jan 2018–Dec 2018	Rohit Kumar
Jan 2018–Dec 2018	Saish Sali
Jan 2018–Dec 2018	Siddesh Shinde
Jan 2018–Dec 2018	Vineeth Ramesh
Aug 2017–Dec 2018	Noopur Anil Maheshwari
Sep 2017–Apr 2018	Tyler Estro
Jun 2017–May 2018	Mayur Jadhav
May 2017–May 2018	Maryia Maskaliova
Jan 2017–Dec 2017	Darshan Godhia
Jan 2017–Dec 2017	Farhaan Jalia
Jan 2017–Dec 2017	Geetika Babu Bangera
Jan 2017–Dec 2017	Kunal Shah
Jan 2017–Dec 2017	Nehil Shah
Jan 2017–Dec 2017	Nidhi Panpalia

Jan 2017–Dec 2017	Rushabh Shah
Jan 2017–Dec 2017	Sagar Shah
Jan 2017–Dec 2017	Swaminathan Sivaraman
Jan 2017–Dec 2017	Tushar Jain
Sep 2016–Dec 2017	Vinothkumar Raja
Aug 2016–Dec 2017	Arun Ramachandran
Aug 2016–Dec 2017	Mukul Sharma
Aug 2016–Dec 2017	Sachin Tiwari
Aug 2016–Dec 2017	Shivanshu Goswami
Aug 2016–Dec 2016	Vishal Sahu
Apr 2016–May 2017	Leixiang Wu
Jan 2016–Dec 2016	Praveen Kumar Morampudi
Jan 2016–Dec 2016	Vithiya Muthukumar
Jan 2016–Dec 2016	Harshkumar Patel
Jan 2016–Dec 2016	Deepika Peringanji
Jan 2016–Dec 2016	Venkatakrishnan Rajagopalan
Jan 2016–Dec 2016	Vishnu Vardhan Rajula
Jan 2016–Dec 2016	Hari Prasath Raman
Sep 2015–Dec 2016	Shubhi Rani
Jan 2016–Dec 2016	Jasmit Saluja
Sep 2015–Dec 2016	Ashok Sankar Harihara Subramony
Aug 2015–Dec 2016	Bharath Kumar Reddy Vangoor
Sep 2014–May 2016	Dongju Ok
Mar 2015–May 2016	Garima Gehlot
Sep 2015–Dec 2015	Vivek Tiwari
May 2014–Dec 2015	Kumar Sourav
Sep 2014–Dec 2015	Bharat Singh
Sep 2014–Dec 2015	Varun Shastry
May 2014–Dec 2015	Abhishek Gupta
Sep 2014–Dec 2015	Arvind Chaudhary
Sep 2014–Dec 2015	Aashray Arora
Sep 2013–May 2015	Arun Olappamanna Vasudevan

Jan 2014–May 2015	Akhilesh Chaganti
May 2014–May 2015	Udit Kaushik Chitalia
Apr 2014–May 2015	Kelong Wang
Sep 2013–Dec 2014	Soujanya Shankaranarayana
May 2014–Dec 2014	Li Mengyang
Sep 2013–May 2014	Amanpreet Mukker
Jun 2013–May 2014	Sonam Mandal
Sep 2012–Dec 2013	Deepak Jain
Sep 2012–Dec 2013	Sagar Trehan
Sep 2012–Dec 2013	Madhu Srikar Palmur
Sep 2012–May 2013	Benixon Arul Dhas
May 2012–Jun 2013	Karthikeyani Palanisami
Jan 2012–Dec 2012	Binesh Andrews
Jan 2012–Dec 2012	Atul Karmarkar
Sep 2011–Dec 2012	Mandar S. Joshi
Sep 2011–Dec 2012	Ravikant R. Malpani
Sep 2011–May 2012	Rajesh Aavuty
Jan 2011–May 2012	Amar Mudrankit
Jan 2011–Dec 2011	Siddhi Tadpatrikar
Sep 2010–Dec 2011	Samriti Katoch
Sep 2010–Dec 2011	Koundinya Santhosh Kumar
Aug 2010–May 2012	Pradeep Shetty
Jan 2011–May 2011	Koundinya Muppalla
Jan 2010–Dec 2010	Prabakar Radhakrishnan
Jan 2010–Dec 2010	Gyumin Sim
Sep 2009–Dec 2010	Shrikar Archak
Sep 2009–Dec 2010	Saumitra Bhanage
Sep 2009–Dec 2010	Sagar Dixit
Jan 2009–May 2010	Priya Sehgal
Jan 2009–Dec 2009	Karthikeyan Srinaivasan
Sep 2008–May 2010	Abhinav Duggal
Sep 2008–Dec 2009	Sujay Godbole

Jan 2008–Dec 2008	Yamini Allu
Jan 2008–Dec 2008	Manjunath Chinni
Jan 2008–Dec 2008	Sachin Gaikwad
Sep 2007–Jun 2009	Daniel J. Dean (domestic)
Sep 2007–Dec 2008	Himanshu Kanda
Sep 2007–May 2009	Rachita Kothiyal
Sep 2007–Dec 2008	Chaitanya Yalamanchili
Jan 2007–Dec 2007	Kimberly Albrecht (domestic, SFS scholar)
Jan 2007–Dec 2007	Ramya Edara
Jan 2007–Dec 2007	Ivan Deras Tabora
Jan 2007–Dec 2007	Kumar Thangavelu
Sep 2006–Dec 2007	Gopala Suryanarayana
Sep 2006–Dec 2007	Kiron Vijayasankar
Jun 2006–Dec 2007	Richard Spillane (domestic)
Jan 2006–August 2007	Swaminathan Sundararaman
Sep 2006–June 2007	Yiannis Pericleous
Sep 2005–May 2007	David Quigley (domestic, SFS scholar)
Jan 2006–March 2007	Harry Papaxenopoulos (domestic, SFS scholar)
Jan 2006–Dec 2006	Chaitanya Patti
Jan 2005–Dec 2005	Dan Ottavio (domestic)
Jan 2005–Dec 2005	Arun Kumar
Sep 2004–Dec 2005	Naveen Gupta
Sep 2004–Dec 2005	Rakesh Narayan Iyer
Sep 2004–Dec 2005	Sunil Satnur
May 2004–Nov 2004	Swapnil V. Patil
Jan 2004–Dec 2004	Mohammad Nayyer Zubair
Sep 2003–Aug 2005	Abhishek Rai
Sep 2003–Dec 2004	Aditya Kashyap
Sep 2003–Dec 2004	Devaki Pandurang Kulkarni
May 2003–Aug 2002	Akshat Khandelwal
May 2003–Aug 2002	Deepak Rao
May 2003–Aug 2004	Akshat Aranya

May 2003–Dec 2003	Alexander Butler (domestic)
May 2003–Dec 2003	Jay Pradip Dave
May 2003–Dec 2003	Mohan-Krishna Channa-Reddy
May 2003–Dec 2003	Salil Gokhale
May 2003–Jan 2004	Abhijith Das
May 2003–Dec 2004	Miretskiy “Eugene” Yevgeniy (domestic)
Jan 2003–Dec 2003	Harikesavan Pathangi Krishnan
Jan 2003–Dec 2003	Puja Gupta
May 2002–Dec 2002	Delia Paval
May 2002–Dec 2002	Nitin Khosla
May 2002–Dec 2002	Rongqing “Frank” Tu
May 2002–May 2003	Jeffrey R. Osborn (domestic)
May 2002–May 2003	Michael Martino (domestic)
May 2002–May 2003	Zhenghong “Sam” Yang
Sep 2002–Dec 2002	Nishant Nagalia
Sep 2002–Dec 2002	Zhou Zhang
Sep 2002–Dec 2003	Swaroop Karunakara
Sep 2002–May 2003	Ana Centeno
Sep 2002–May 2003	Andrew Himmer (domestic)
Sep 2002–May 2003	Sheshadri Sreenath
Mar 2002–Apr 2003	Manish Prasad
Jan 2002–May 2004	Kiran-Kumar Muniswamy-Reddy
Sep 2001–May 2003	Amit Purohit

UNDERGRADUATE STUDENTS

Jun 2021–present	Michael Arkhangelskiy
Jun 2021–present	Andrew Burford
Jun 2021–present	Michael McNeill

Past Undergraduate Students

Jun 2021–Jun 2022	Pei Liu (domestic, BS program, CSE program)
Jun 2020–Dec 2020	Haolin Yu (BS program, CSE program)
Jun 2020–May 2021	Haewon Julie Lee (BS program, CSE program)
Jun 2020–May 2021	Yang Yang (domestic, BS program, CSE program)

Aug 2019–May 2020	Yinuo Zhang (BS program, CSE program)
Jul 2019–Dec 2019	Abraham Spitalny, (domestic, BS program, CSE program)
May 2018–Dec 2018	Amrith Arunachalam (domestic, BS program, CSE program)
May 2017–May 2018	Kevin Sun (domestic, BS program, CSE program)
Feb 2015–Mar 2016	Leixiang Wu (domestic, BS program, CSE program)
July 2012–Dec 2012	Edgardo Linero, (domestic, BS program, CSE program)
May 2011–May 2012	Ryan Matthew Walsh, (domestic, BS program, ISE program)
May 2010–Aug 2011	Christopher Jutting, (domestic, BS program, ISE program)
Jan 2009–July 2009	Ryan Leif Walsh, (domestic, BS/MS program, Honors program)
Jan 2004–2009	Adam David Alan Martin (domestic)
Sep 2005–May 2008	Josef "Jeff" Sipek (domestic)
Jun 2006–August 2007	Daniel J. Dean (domestic)
Jan 2005–May 2006	Richard Spillane (domestic, BS/MS program, Honors program, Stony Brook URECA summer internship)
Jan 2005–Jul 2005	Tim Wong (domestic)
Jan 2005–Aug 2005	David Quigley (domestic)
May 2003–Dec 2003	Mohammad Nayyer Zubair (Honors program, Stony Brook URECA summer internship)
Jan 2002–May 2002	Evan Chan (domestic)
Jan 2002–May 2002	Reynold Mercado (domestic)
Jan 2002–May 2002	Jeffrey Osborn (domestic)
Jan 2002–May 2002	Julio Salazar (domestic)
Jan 2002–May 2002	Ariye Shater (domestic)
Jan 2002–May 2003	Charles P. Wright (Honors program advisor, domestic)
Jan 2002–May 2002	Yuri Yanpolski (domestic)

HIGH-SCHOOL STUDENTS

Past High-School Students

Sep 2015–Aug 2017	Henry Nelson (domestic), "NFSv4 Compounds (vNFS)" project
June 2005–August 2005	Jordan Hoch (domestic), Simons summer internship, project title "Visual Debugging of Operating Systems State.")

Stony Brook, New York, April 30, 2025