US 20050235092A1

(19) United States

(12) Patent Application Publication

(10) Pub. No.: US 2005/0235092 A1

(43) Pub. Date: Oct. 20, 2005

Ballew et al.

(54) HIGH PERFORMANCE COMPUTING SYSTEM AND METHOD

(75) Inventors: **James D. Ballew**, Grapevine, TX (US); **Gary R. Early**, Allen, TX (US)

Correspondence Address:
BAKER BOTTS LLP
2001 ROSS AVENUE
6TH FLOOR
DALLAS, TX 75201 (US)

(73) Assignee: **Raytheon Company**

(21) Appl. No.: **10/824,874**

(22) Filed: **Apr. 15, 2004**

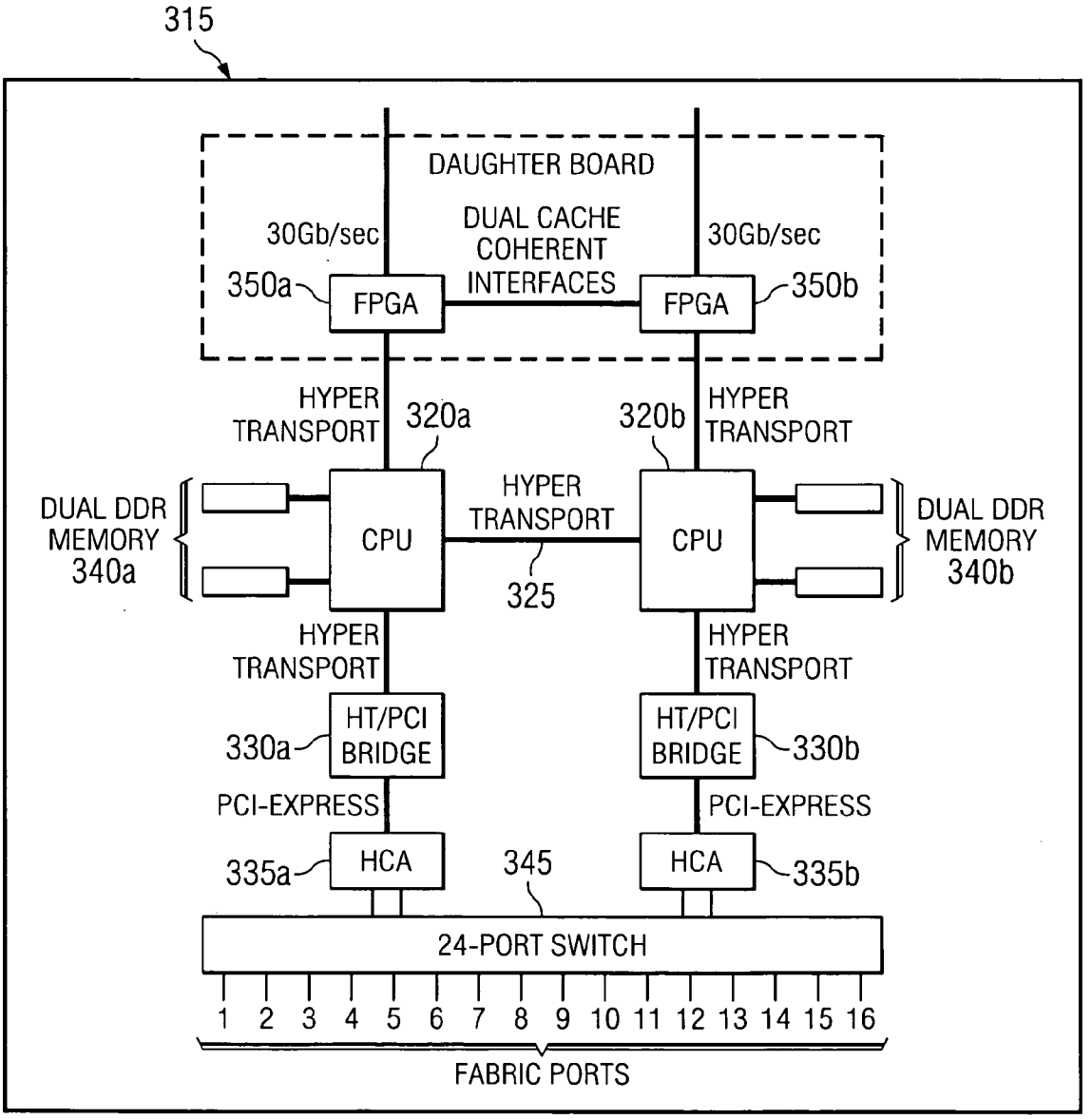
Publication Classification

(51) **Int. Cl.**[7] **G06F 13/00**

(52) **U.S. Cl.** **710/316**; 710/306

(57) **ABSTRACT**

A High Performance Computing (HPC) node comprises a motherboard, a switch comprising eight or more ports integrated on the motherboard, and at least two processors operable to execute an HPC job, with each processor communicably coupled to the integrated switch and integrated on the motherboard.



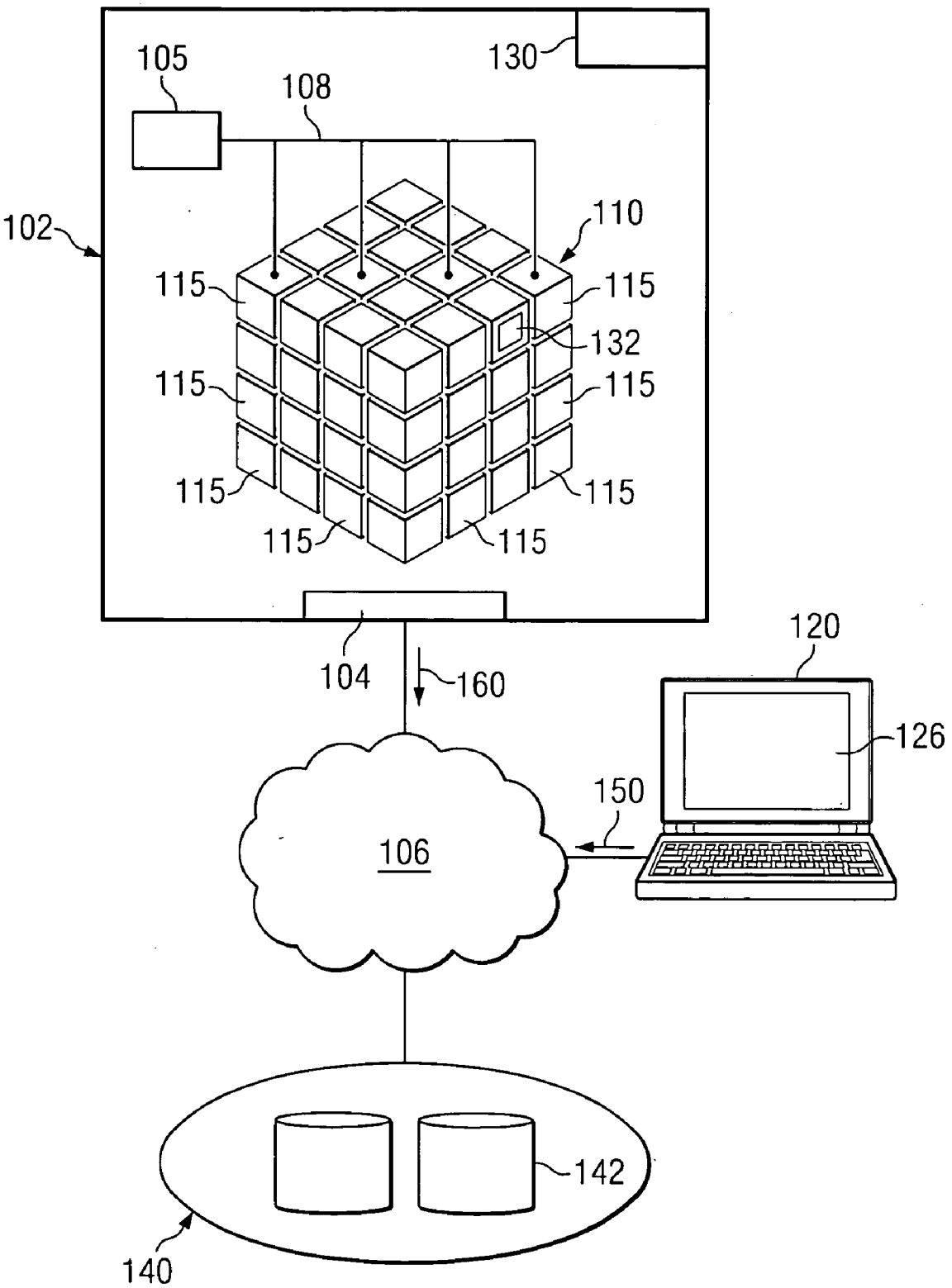

105
108
130
102
110
115
115
132
115
115
115
115
115
115
104
160
120
126
150
106
142
140

FIG. 1

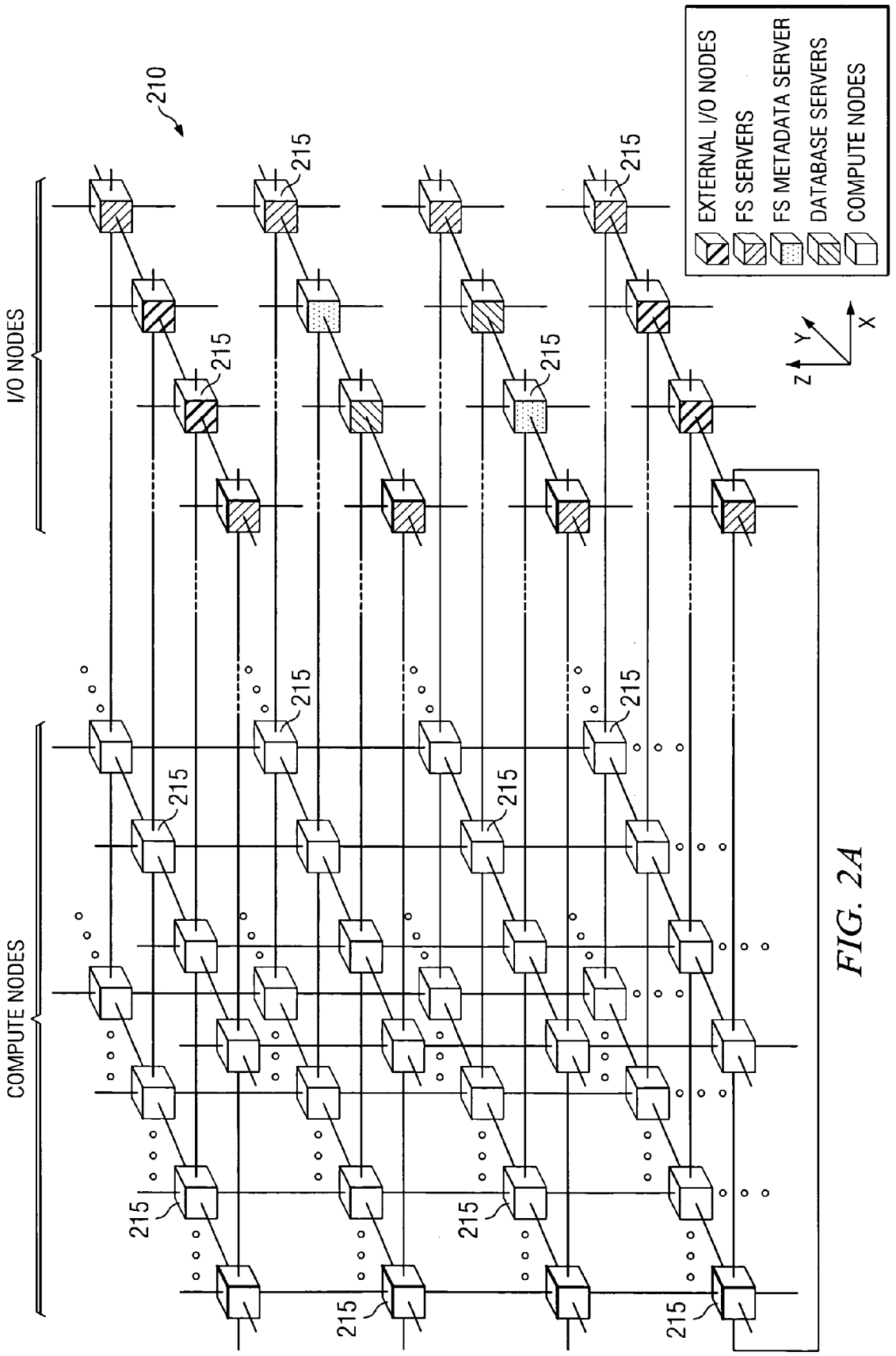

210
I/O NODES
COMPUTE NODES
215
EXTERNAL I/O NODES
FS SERVERS
FS METADATA SERVER
DATABASE SERVERS
COMPUTE NODES
X
Y
Z

FIG. 2A

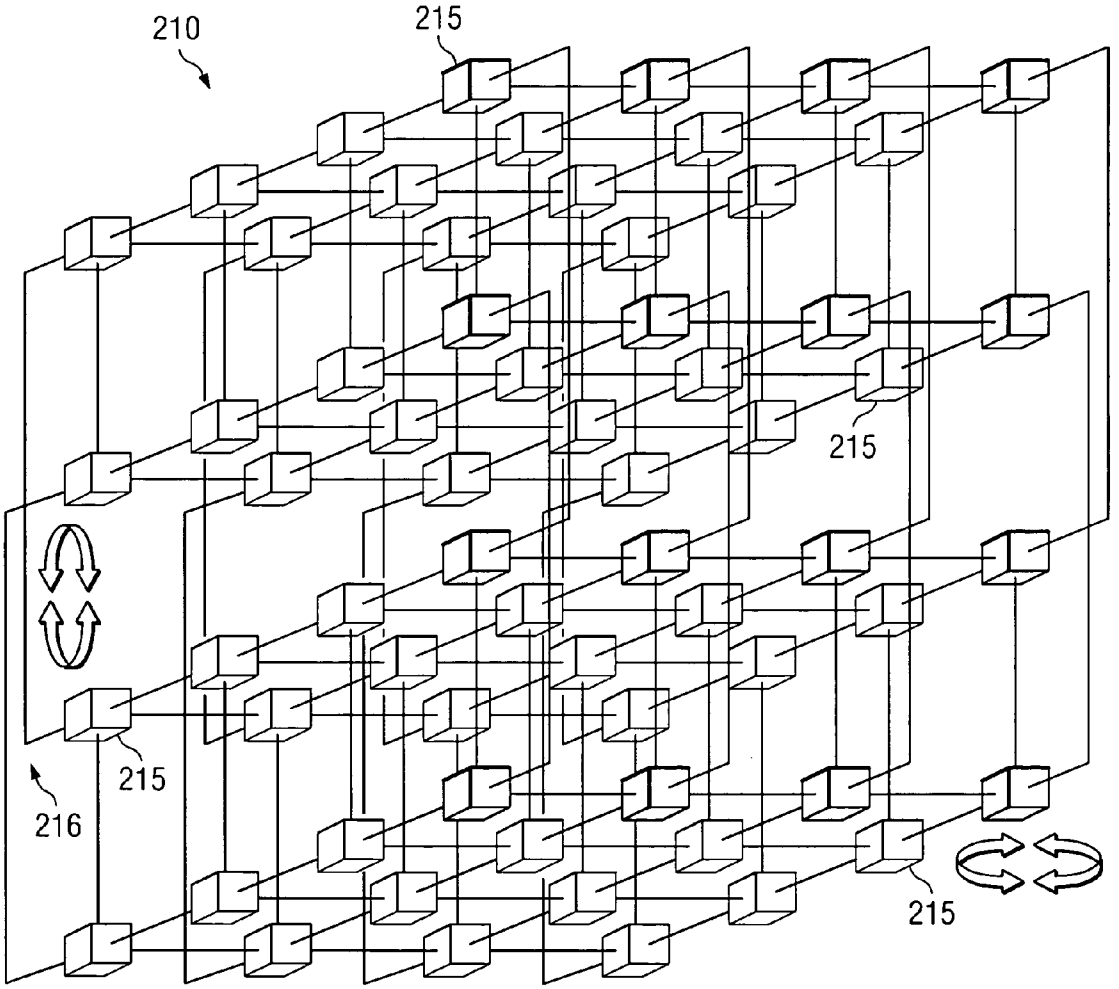


FIG. 2B

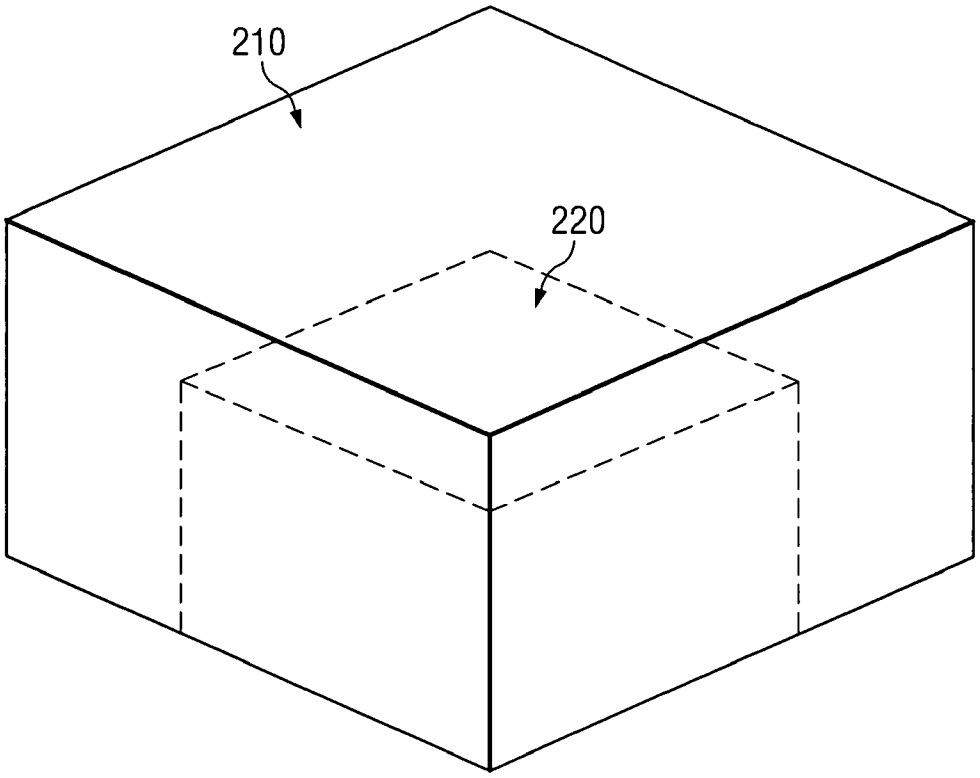


FIG. 2C

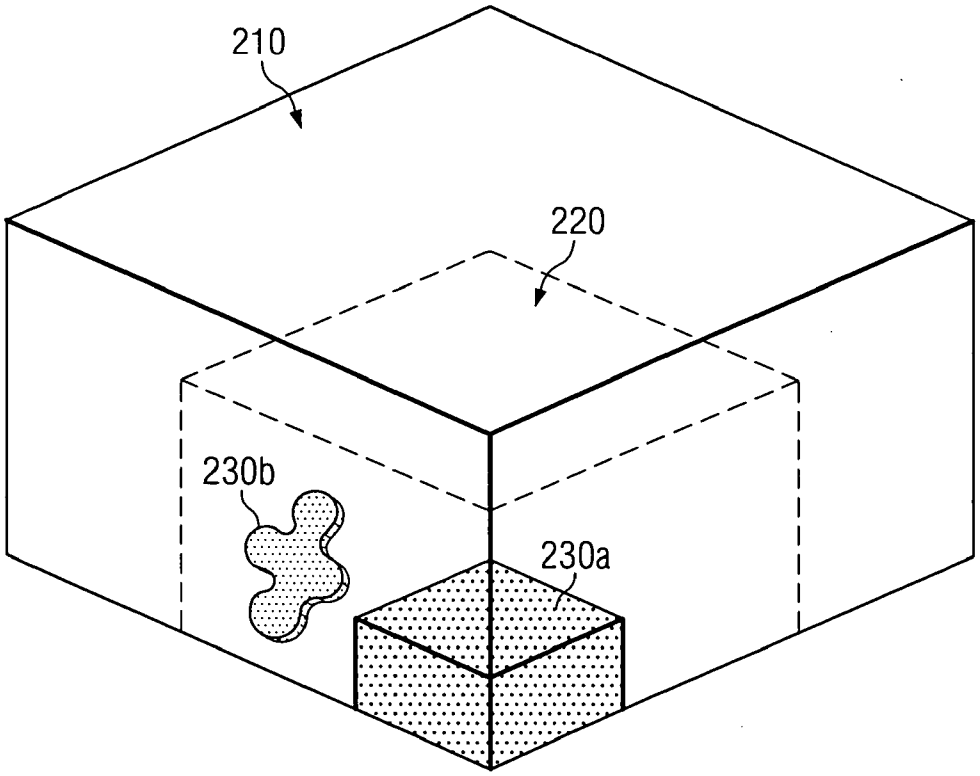


FIG. 2D

FIG. 3A

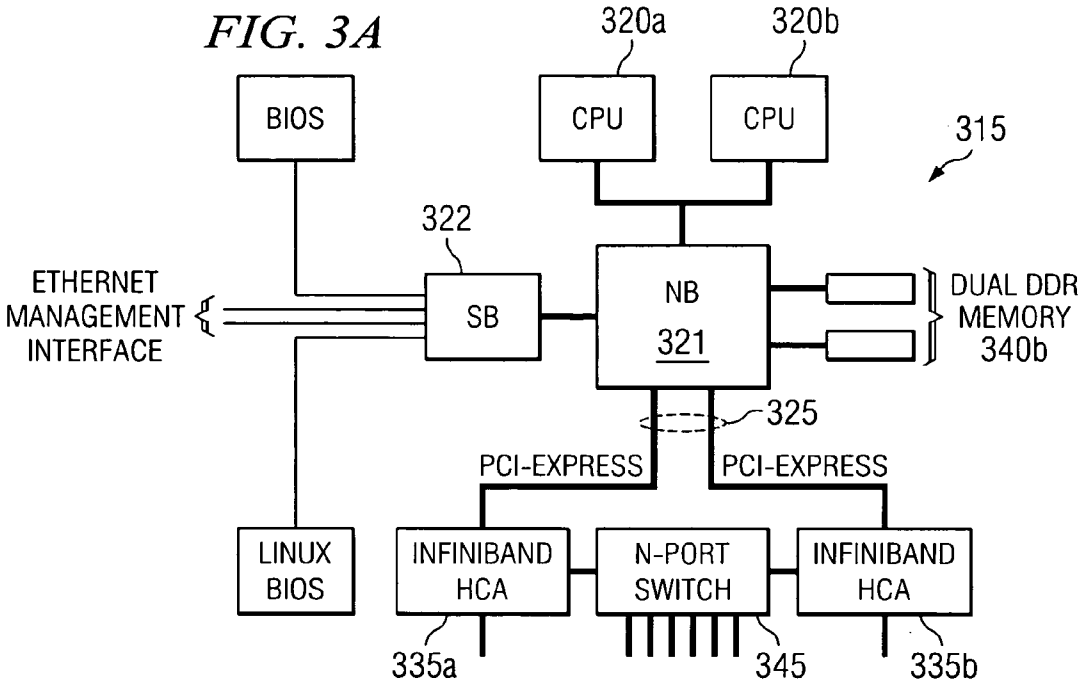

320a
320b
BIOS
CPU
CPU
315
322
ETHERNET
MANAGEMENT
INTERFACE
SB
NB
321
DUAL DDR
MEMORY
340b
325
PCI-EXPRESS
PCI-EXPRESS
LINUX
BIOS
INFINIBAND
HCA
N-PORT
SWITCH
INFINIBAND
HCA
335a
345
335b

FIG. 3B

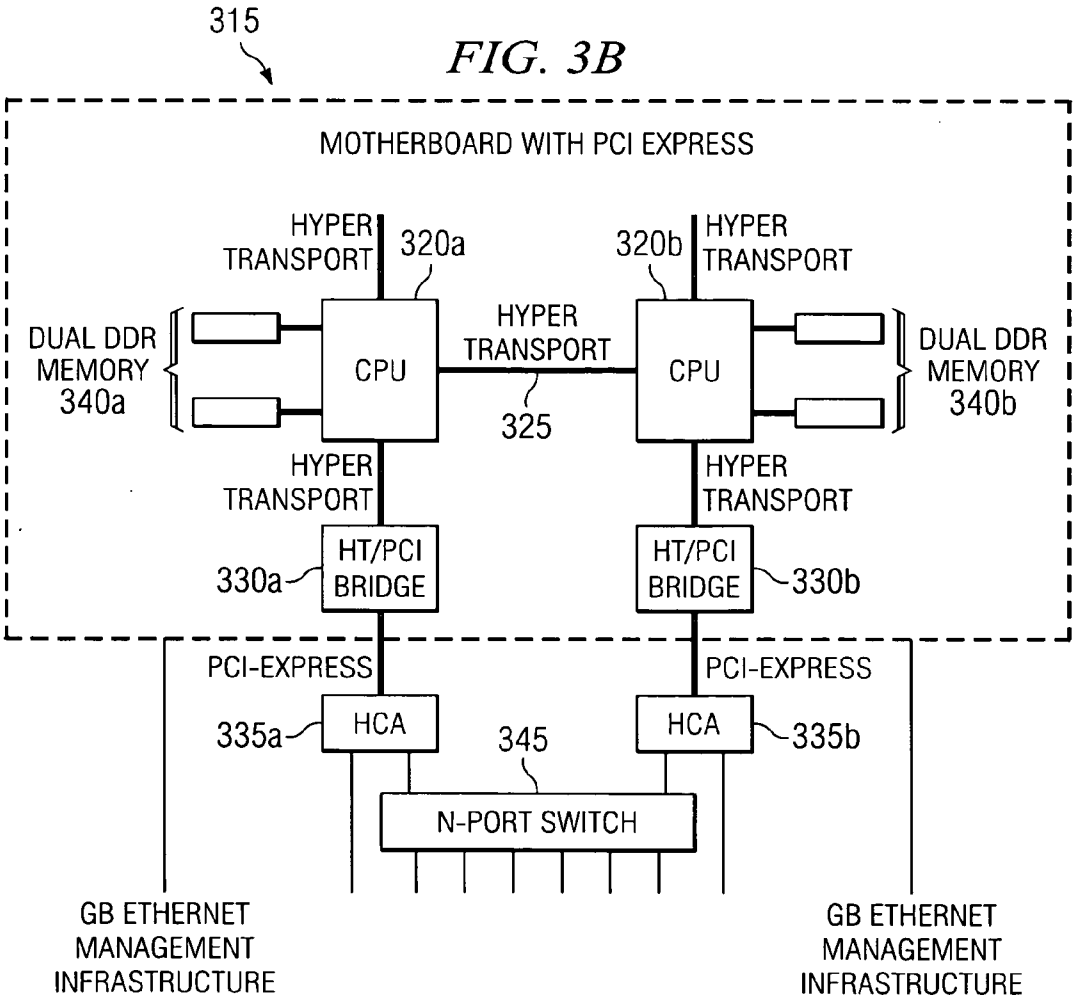
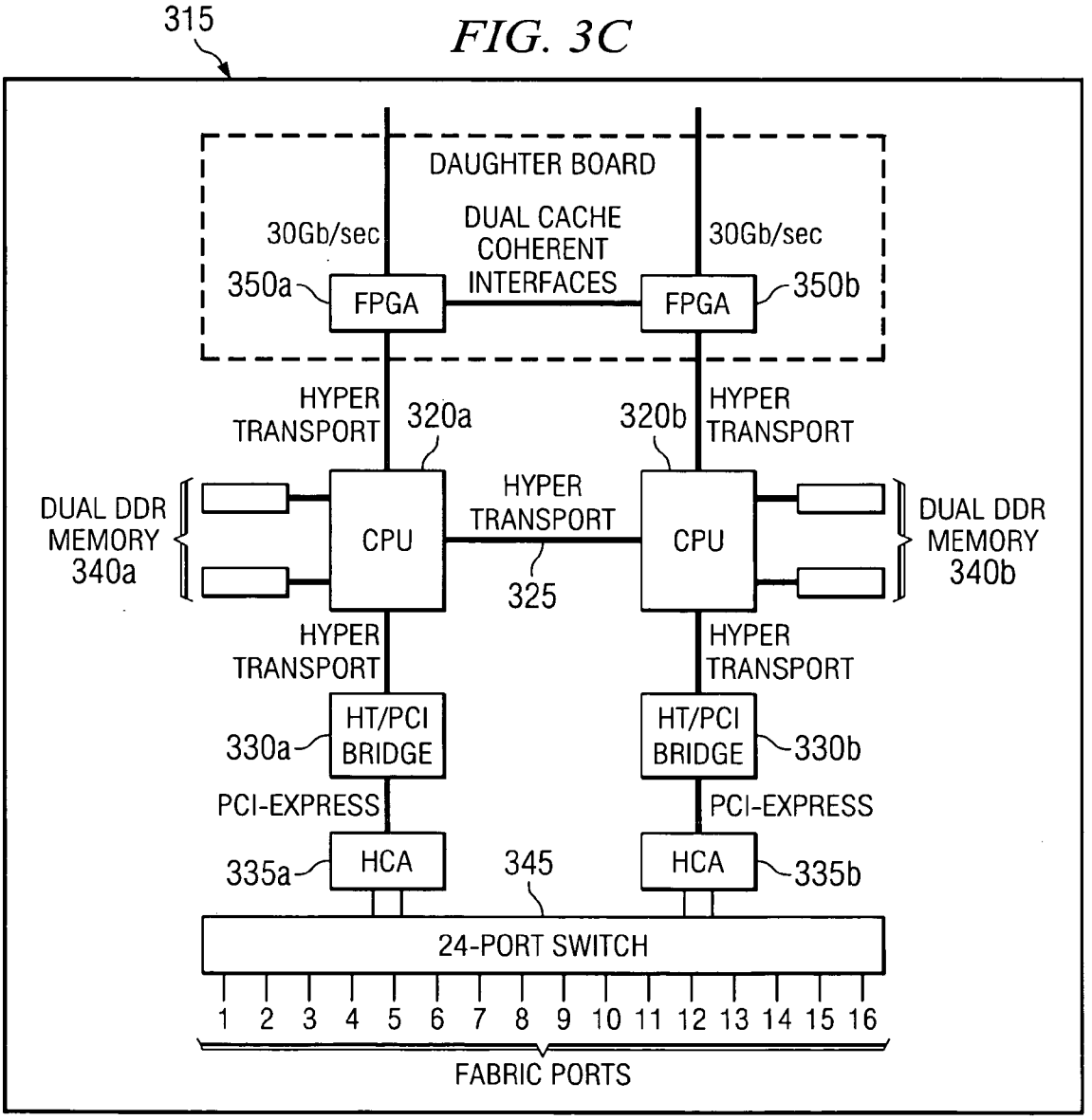

315
MOTHERBOARD WITH PCI EXPRESS
HYPER
TRANSPORT
320a
320b
HYPER
TRANSPORT
DUAL DDR
MEMORY
340a
CPU
HYPER
TRANSPORT
CPU
DUAL DDR
MEMORY
340b
325
HYPER
TRANSPORT
HYPER
TRANSPORT
HT/PCI
BRIDGE
330a
HT/PCI
BRIDGE
330b
PCI-EXPRESS
PCI-EXPRESS
335a
HCA
345
HCA
335b
N-PORT SWITCH
GB ETHERNET
MANAGEMENT
INFRASTRUCTURE
GB ETHERNET
MANAGEMENT
INFRASTRUCTURE

FIG. 3C


315
DAUGHTER BOARD
DUAL CACHE COHERENT INTERFACES
30Gb/sec
30Gb/sec
350a
FPGA
FPGA
350b
HYPER TRANSPORT
320a
320b
HYPER TRANSPORT
DUAL DDR MEMORY 340a
CPU
HYPER TRANSPORT
CPU
DUAL DDR MEMORY 340b
325
HYPER TRANSPORT
HYPER TRANSPORT
330a
HT/PCI BRIDGE
HT/PCI BRIDGE
330b
PCI-EXPRESS
PCI-EXPRESS
335a
HCA
345
HCA
335b
24-PORT SWITCH
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
FABRIC PORTS

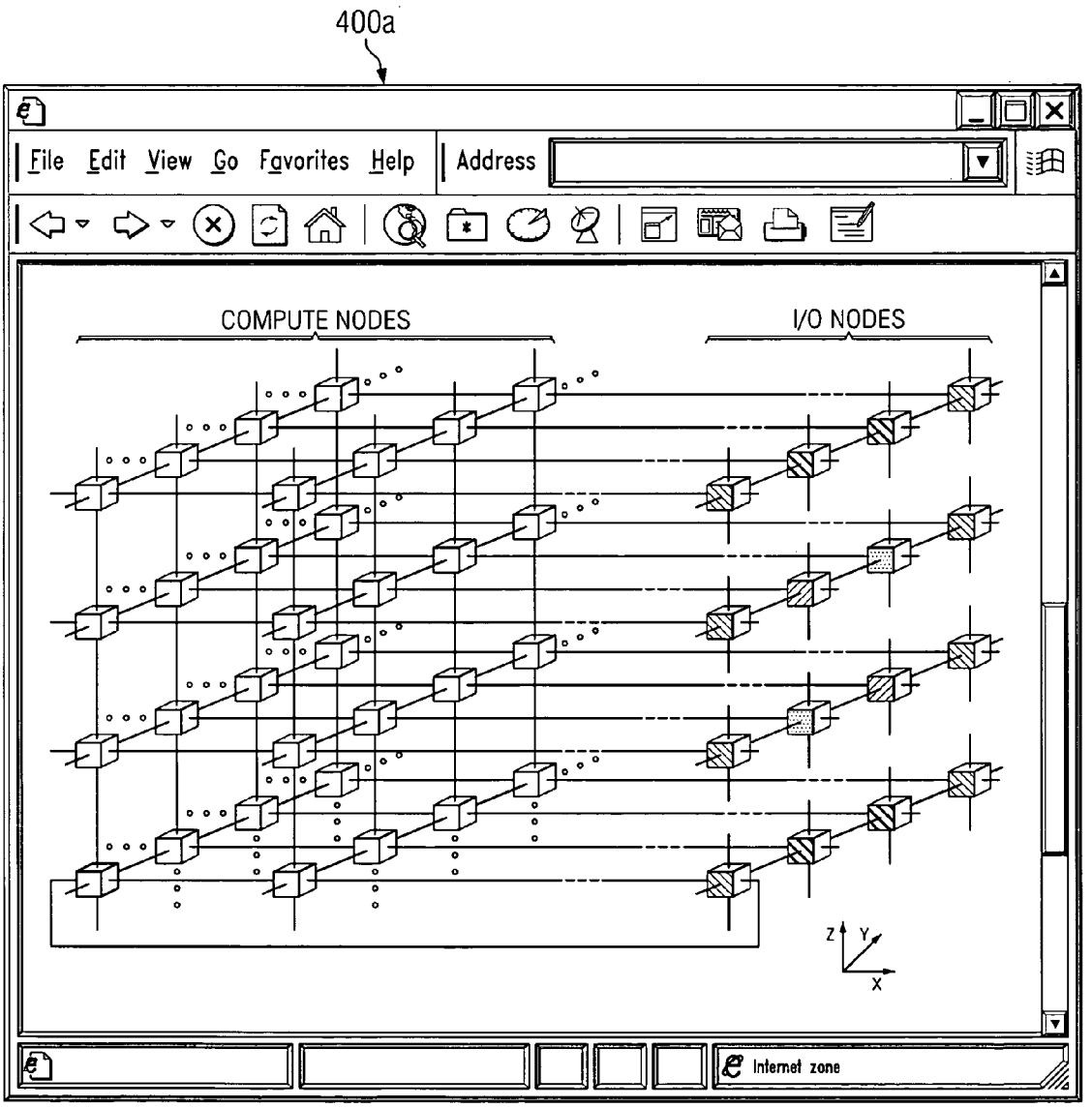

400a
File Edit View Go Favorites Help
Address
COMPUTE NODES
I/O NODES
Z
Y
X
Internet zone

FIG. 4A

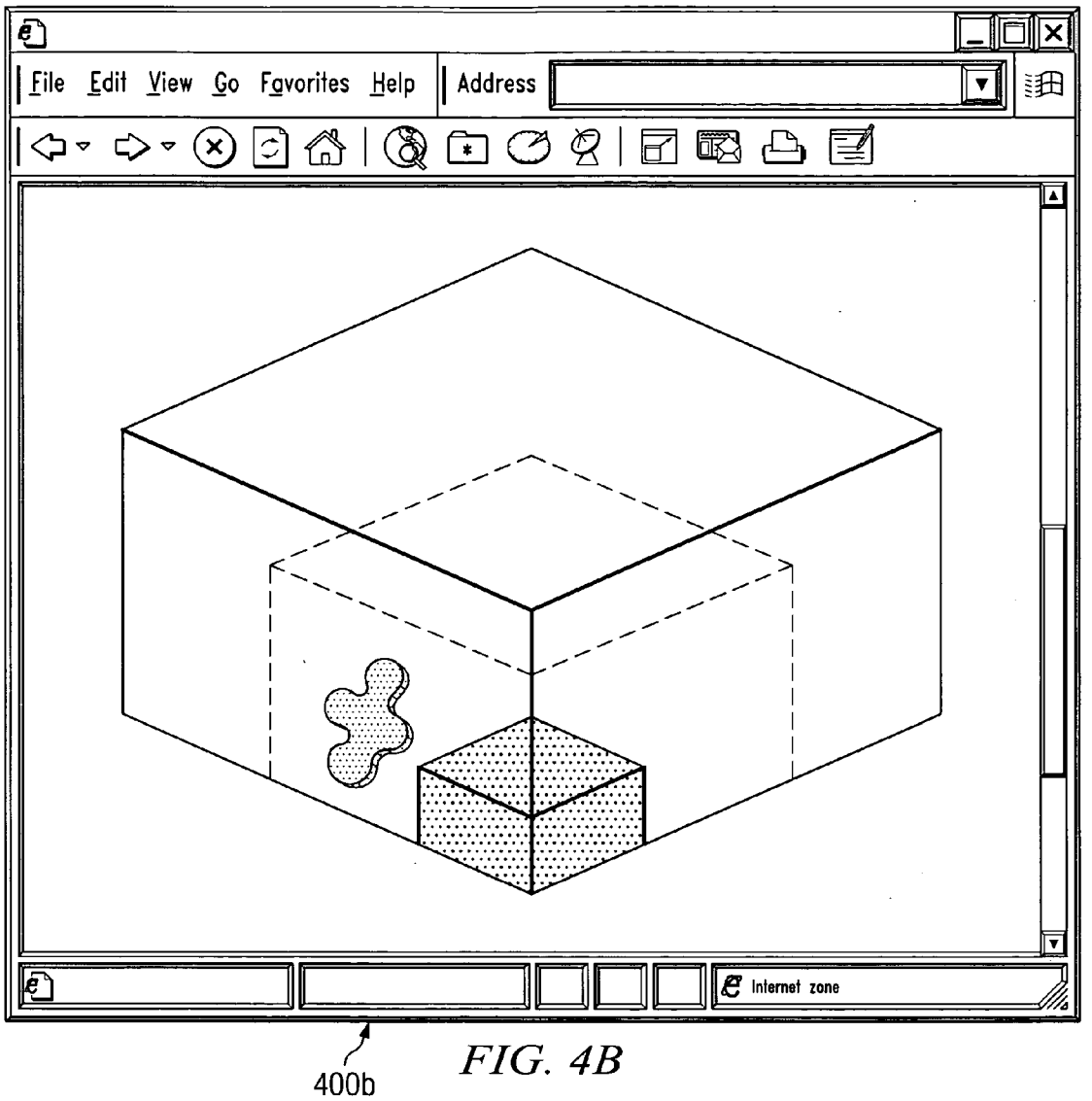

File Edit View Go Favorites Help
Address
Internet zone
400b

FIG. 4B

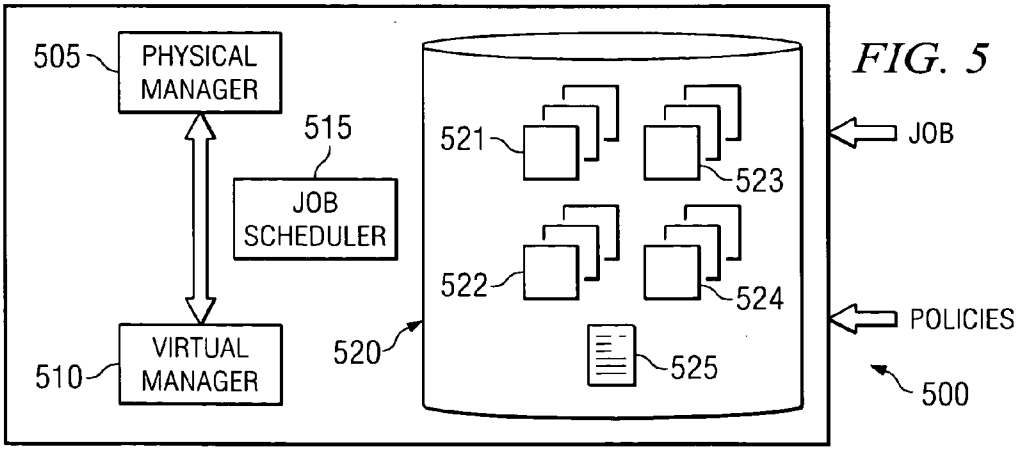
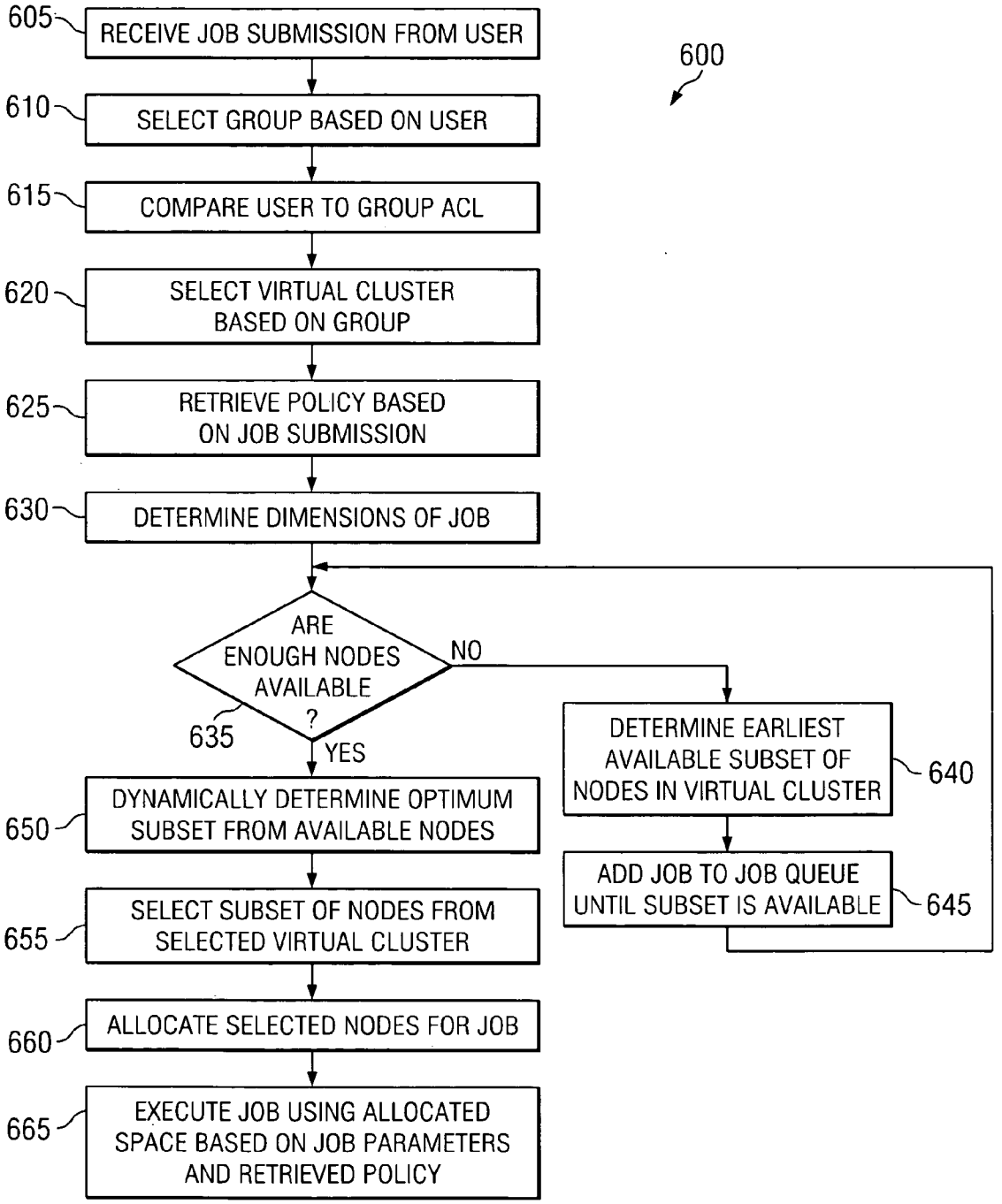
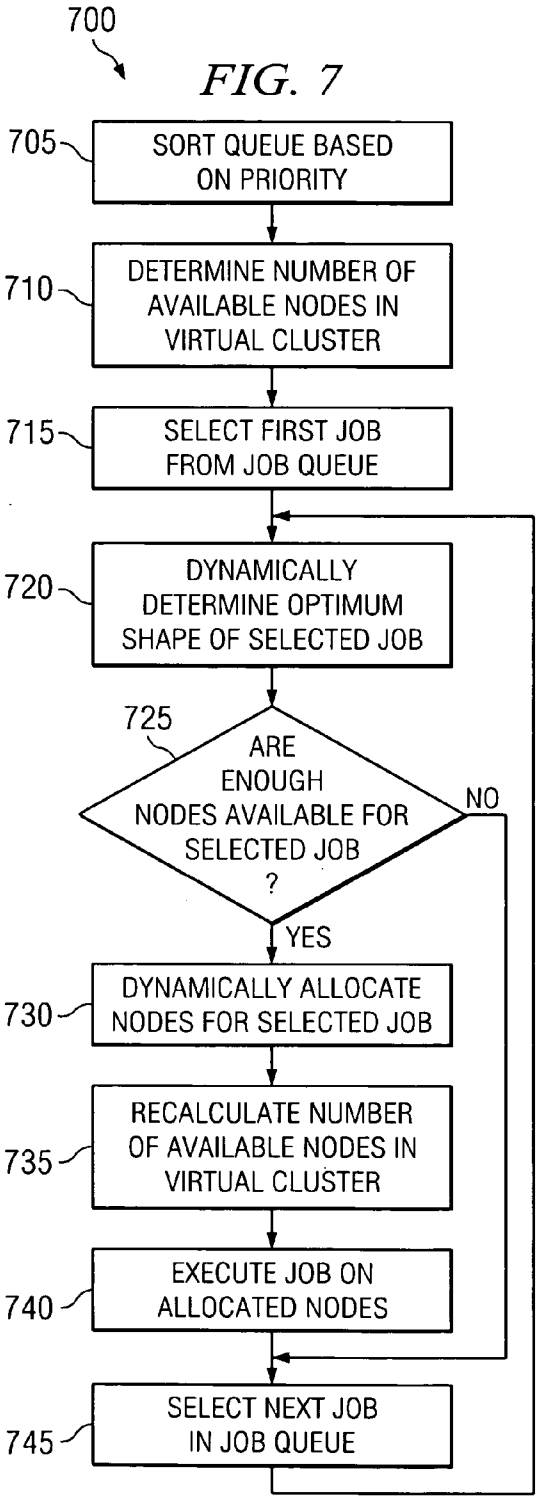
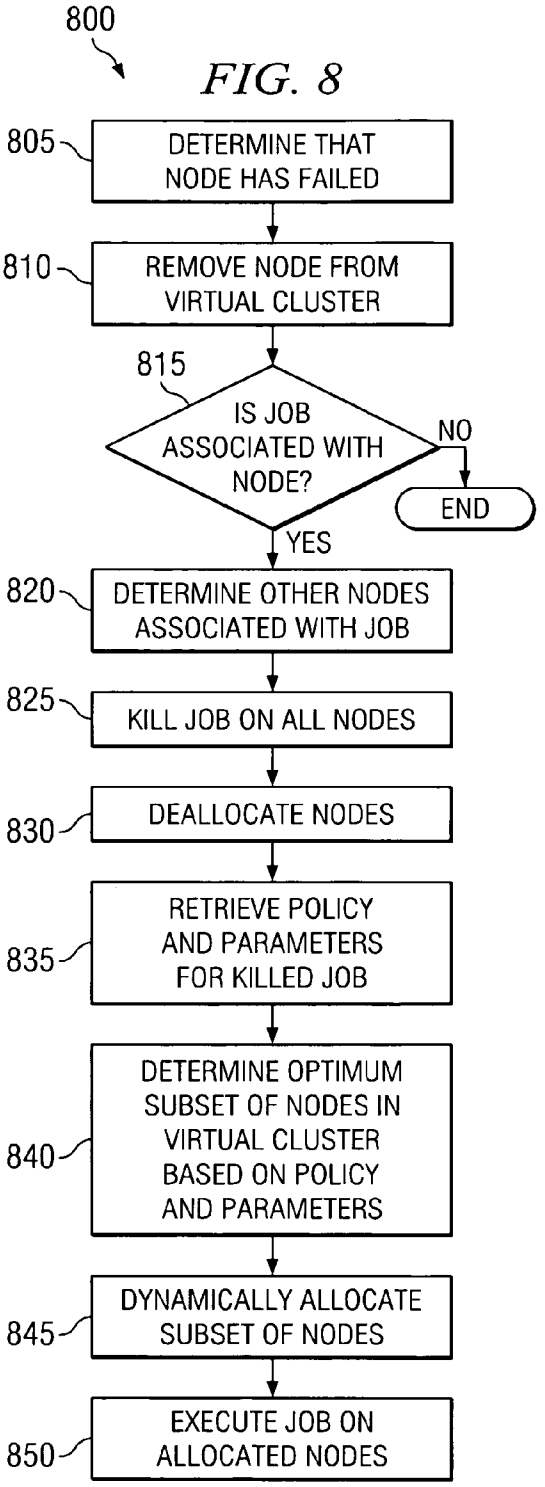

505
PHYSICAL MANAGER
515
JOB SCHEDULER
510
VIRTUAL MANAGER
520
521
522
523
524
525
JOB
POLICIES
500

FIG. 5

FIG. 6


600
605
RECEIVE JOB SUBMISSION FROM USER
610
SELECT GROUP BASED ON USER
615
COMPARE USER TO GROUP ACL
620
SELECT VIRTUAL CLUSTER BASED ON GROUP
625
RETRIEVE POLICY BASED ON JOB SUBMISSION
630
DETERMINE DIMENSIONS OF JOB
ARE ENOUGH NODES AVAILABLE ?
635
NO
YES
DETERMINE EARLIEST AVAILABLE SUBSET OF NODES IN VIRTUAL CLUSTER
640
ADD JOB TO JOB QUEUE UNTIL SUBSET IS AVAILABLE
645
650
DYNAMICALLY DETERMINE OPTIMUM SUBSET FROM AVAILABLE NODES
655
SELECT SUBSET OF NODES FROM SELECTED VIRTUAL CLUSTER
660
ALLOCATE SELECTED NODES FOR JOB
665
EXECUTE JOB USING ALLOCATED SPACE BASED ON JOB PARAMETERS AND RETRIEVED POLICY


700
FIG. 7
705
SORT QUEUE BASED ON PRIORITY
710
DETERMINE NUMBER OF AVAILABLE NODES IN VIRTUAL CLUSTER
715
SELECT FIRST JOB FROM JOB QUEUE
720
DYNAMICALLY DETERMINE OPTIMUM SHAPE OF SELECTED JOB
725
ARE ENOUGH NODES AVAILABLE FOR SELECTED JOB ?
NO
YES
730
DYNAMICALLY ALLOCATE NODES FOR SELECTED JOB
735
RECALCULATE NUMBER OF AVAILABLE NODES IN VIRTUAL CLUSTER
740
EXECUTE JOB ON ALLOCATED NODES
745
SELECT NEXT JOB IN JOB QUEUE


800
FIG. 8
805
DETERMINE THAT NODE HAS FAILED
810
REMOVE NODE FROM VIRTUAL CLUSTER
815
IS JOB ASSOCIATED WITH NODE?
NO
END
YES
820
DETERMINE OTHER NODES ASSOCIATED WITH JOB
825
KILL JOB ON ALL NODES
830
DEALLOCATE NODES
835
RETRIEVE POLICY AND PARAMETERS FOR KILLED JOB
840
DETERMINE OPTIMUM SUBSET OF NODES IN VIRTUAL CLUSTER BASED ON POLICY AND PARAMETERS
845
DYNAMICALLY ALLOCATE SUBSET OF NODES
850
EXECUTE JOB ON ALLOCATED NODES

HIGH PERFORMANCE COMPUTING SYSTEM AND METHOD

TECHNICAL FIELD

[0001] This disclosure relates generally to the field of data processing and, more specifically, to a high performance computing system and method.

BACKGROUND OF THE INVENTION

[0002] High Performance Computing (HPC) is often characterized by the computing systems used by scientists and engineers for modeling, simulating, and analyzing complex physical or algorithmic phenomena. Currently, HPC machines are typically designed using numerous HPC clusters of one or more processors referred to as nodes. For most large scientific and engineering applications, performance is chiefly determined by parallel scalability and not the speed of individual nodes; therefore, scalability is often a limiting factor in building or purchasing such high performance clusters. Scalability is generally considered to be based on i) hardware, ii) memory, I/O, and communication bandwidth; iii) software; iv) architecture; and v) applications. The processing, memory, and I/O bandwidth in most conventional HPC environments are normally not well balanced and, therefore, do not scale well. Many HPC environments do not have the I/O bandwidth to satisfy high-end data processing requirements or are built with blades that have too many unneeded components installed, which tend to dramatically reduce the system's reliability. Accordingly, many HPC environments may not provide robust cluster management software for efficient operation in production-oriented environments.

SUMMARY OF THE INVENTION

[0003] This disclosure provides a High Performance Computing (HPC) node comprises a motherboard, a switch comprising eight or more ports integrated on the motherboard, and at least two processors operable to execute an HPC job, with each processor communicably coupled to the integrated switch and integrated on the motherboard.

[0004] The invention has several important technical advantages. For example, one possible advantage of the present invention is that by at least partially reducing, distributing, or eliminating centralized switching functionality, it may provide greater input/output (I/O) performance, perhaps four to eight times the conventional HPC bandwidth. Indeed, in certain embodiments, the I/O performance may nearly equal processor performance. This well-balanced approach may be less sensitive to communications overhead. Accordingly, the present invention may increase blade and overall system performance. A further possible advantage is reduced interconnect latency. Further, the present invention may be more easily scaleable, reliable, and fault tolerant than conventional blades. Yet another advantage may be a reduction of the costs involved in manufacturing an HPC server, which may be passed on to universities and engineering labs, and/or the costs involved in performing HPC processing. The invention may further allow for management software that is more robust and efficient based, at least in part, on the balanced architecture. Various embodiments of the invention may have none, some, or all of these advantages. Other technical advantages of the present invention will be readily apparent to one skilled in the art.

BRIEF DESCRIPTION OF THE DRAWINGS

[0005] For a more complete understanding of the present disclosure and its advantages, reference is now made to the following descriptions, taken in conjunction with the accompanying drawings, in which:

[0006] **FIG. 1** illustrates an example high-performance computing system in accordance with one embodiment of the present disclosure;

[0007] FIGS. **2**A-D illustrate various embodiments of the grid in the system of **FIG. 1** and the usage thereof;

[0008] FIGS. **3**A-C illustrate various embodiments of individual nodes in the system of **FIGS. 1**;

[0009] FIGS. **4**A-B illustrate various embodiments of a graphical user interface in accordance with the system of **FIG. 1**;

[0010] **FIG. 5** illustrates one embodiment of the cluster management software in accordance with the system in **FIG. 1**;

[0011] **FIG. 6** is a flowchart illustrating a method for submitting a batch job in accordance with the high-performance computing system of **FIG. 1**;

[0012] **FIG. 7** is a flowchart illustrating a method for dynamic backfilling of the grid in accordance with the high-performance computing system of **FIG. 1**; and

[0013] **FIG. 8** is a flow chart illustrating a method for dynamically managing a node failure in accordance with the high-performance computing system of **FIG. 1**.

DETAILED DESCRIPTION OF THE DRAWINGS

[0014] **FIG. 1** is a block diagram illustrating a high Performance Computing (HPC) system **100** for executing software applications and processes, for example an atmospheric, weather, or crash simulation, using HPC techniques. System **100** provides users with HPC functionality dynamically allocated among various computing nodes **115** with I/O performance substantially similar to the processing performance. Generally, these nodes **115** are easily scaleable because of, among other things, this increased input/output (I/O) performance and reduced fabric latency. For example, the scalability of nodes **115** in a distributed architecture may be represented by a derivative of Amdahl's law:

$$S(N)=1/((FP/N)+FS)*(1-Fc*(1-RR/L))$$

[0015] where S(N)=Speedup on N processors, Fp=Fraction of Parallel Code, Fs=Fraction of Non-Parallel Code, Fc=Fraction of processing devoted to communications, and RR/L=Ratio of Remote/Local Memory Bandwidth. Therefore, by HPC system **100** providing I/O performance substantially equal to or nearing processing performance, HPC system **100** increases overall efficiency of HPC applications and allows for easier system administration.

[0016] HPC system **100** is a distributed client/server system that allows users (such as scientists and engineers) to submit jobs **150** for processing on an HPC server **102**. For example, system **100** may include HPC server **102** that is connected, through network **106**, to one or more administration workstations or local clients **120**. But system **100** may be a standalone computing environment or any other suitable environment. In short, system **100** is any HPC

computing environment that includes highly scaleable nodes **115** and allows the user to submit jobs **150**, dynamically allocates scaleable nodes **115** for job **150**, and automatically executes job **150** using the allocated nodes **115**. Job **150** may be any batch or online job operable to be processed using HPC techniques and submitted by any apt user. For example, job **150** may be a request for a simulation, a model, or for any other high-performance requirement. Job **150** may also be a request to run a data center application, such as a clustered database, an online transaction processing system, or a clustered application server. The term "dynamically," as used herein, generally means that certain processing is determined, at least in part, at run-time based on one or more variables. The term "automatically," as used herein, generally means that the appropriate processing is substantially performed by at least part of HPC system **100**. It should be understood that "automatically" further contemplates any suitable user or administrator interaction with system **100** without departing from the scope of this disclosure.

[**0017**] HPC server **102** comprises any local or remote computer operable to process job **150** using a plurality of balanced nodes **115** and cluster management engine **130**. Generally, HPC server **102** comprises a distributed computer such as a blade server or other distributed server. However the configuration, server **102** includes a plurality of nodes **115**. Nodes **115** comprise any computer or processing device such as, for example, blades, general-purpose personal computers (PC), Macintoshes, workstations, Unix-based computers, or any other suitable devices. Generally, **FIG. 1** provides merely one example of computers that may be used with the disclosure. For example, although **FIG. 1** illustrates one server **102** that may be used with the disclosure, system **100** can be implemented using computers other than servers, as well as a server pool. In other words, the present disclosure contemplates computers other than general purpose computers as well as computers without conventional operating systems. As used in this document, the term "computer" is intended to encompass a personal computer, workstation, network computer, or any other suitable processing device. HPC server **102**, or the component nodes **115**, may be adapted to execute any operating system including Linux, UNIX, Windows Server, or any other suitable operating system. According to one embodiment, HPC server **102** may also include or be communicably coupled with a remote web server. Therefore, server **102** may comprise any computer with software and/or hardware in any combination suitable to dynamically allocate nodes **115** to process HPC job **150**.

[**0018**] At a high level, HPC server **102** includes a management node **105**, a grid **110** comprising a plurality of nodes **115**, and cluster management engine **130**. More specifically, server **102** may be a standard 19" rack including a plurality of blades (nodes **115**) with some or all of the following components: i) dual-processors; ii) large, high bandwidth memory; iii) dual host channel adapters (HCAs); iv) integrated fabric switching; v) FPGA support; and vi) redundant power inputs or N+1 power supplies. These various components allow for failures to be confined to the node level. But it will be understood that HPC server **102** and nodes **115** may not include all of these components.

[**0019**] Management node **105** comprises at least one blade substantially dedicated to managing or assisting an administrator. For example, management node **105** may comprise two blades, with one of the two blades being redundant (such as an active/passive configuration). In one embodiment, management node **105** may be the same type of blade or computing device as HPC nodes **115**. But, management node **105** may be any node, including any number of circuits and configured in any suitable fashion, so long as it remains operable to at least partially manage grid **110**. Often, management node **105** is physically or logically separated from the plurality of HPC nodes **115**, jointly represented in grid **110**. In the illustrated embodiment, management node **105** may be communicably coupled to grid **110** via link **108**. Link **108** may comprise any communication conduit implementing any appropriate communications protocol. In one embodiment, link **108** provides Gigabit or 10 Gigabit Ethernet communications between management node **105** and grid **110**.

[**0020**] Grid **110** is a group of nodes **115** interconnected for increased processing power. Typically, grid **110** is a 3D Torus, but it may be a mesh, a hypercube, or any other shape or configuration without departing from the scope of this disclosure. The links between nodes **115** in grid **110** may be serial or parallel analog links, digital links, or any other type of link that can convey electrical or electromagnetic signals such as, for example, fiber or copper. Each node **115** is configured with an integrated switch. This allows node **115** to more easily be the basic construct for the 3D Torus and helps minimize XYZ distances between other nodes **115**. Further, this may make copper wiring work in larger systems at up to Gigabit rates with, in some embodiments, the longest cable being less than 5 meters. In short, node **115** is generally optimized for nearest-neighbor communications and increased I/O bandwidth.

[**0021**] Each node **115** may include a cluster agent **132** communicably coupled with cluster management engine **130**. Generally, agent **132** receives requests or commands from management node **105** and/or cluster management engine **130**. Agent **132** could include any hardware, software, firmware, or combination thereof operable to determine the physical status of node **115** and communicate the processed data, such as through a "heartbeat," to management node **105**. In another embodiment, management node **105** may periodically poll agent **132** to determine the status of the associated node **115**. Agent **132** may be written in any appropriate computer language such as, for example, C, C++, Assembler, Java, Visual Basic, and others or any combination thereof so long as it remains compatible with at least a portion of cluster management engine **130**.

[**0022**] Cluster management engine **130** could include any hardware, software, firmware, or combination thereof operable to dynamically allocate and manage nodes **115** and execute job **150** using nodes **115**. For example, cluster management engine **130** may be written or described in any appropriate computer language including C, C++, Java, Visual Basic, assembler, any suitable version of 4GL, and others or any combination thereof. It will be understood that while cluster management engine **130** is illustrated in **FIG. 1** as a single multi-tasked module, the features and functionality performed by this engine may be performed by multiple modules such as, for example, a physical layer module, a virtual layer module, a job scheduler, and a presentation engine (as shown in more detail in **FIG. 5**). Further, while illustrated as external to management node **105**, management node **105** typically executes one or more processes associated with cluster management engine **130**

and may store cluster management engine **130**. Moreover, cluster management engine **130** may be a child or sub-module of another software module without departing from the scope of this disclosure. Therefore, cluster management engine **130** comprises one or more software modules operable to intelligently manage nodes **115** and jobs **150**.

[**0023**] Server **102** may include interface **104** for communicating with other computer systems, such as client **120**, over network **106** in a client-server or other distributed environment. In certain embodiments, server **102** receives jobs **150** or job policies from network **106** for storage in disk farm **140**. Disk farm **140** may also be attached directly to the computational array using the same wideband interfaces that interconnects the nodes. Generally, interface **104** comprises logic encoded in software and/or hardware in a suitable combination and operable to communicate with network **106**. More specifically, interface **104** may comprise software supporting one or more communications protocols associated with communications network **106** or hardware operable to communicate physical signals.

[**0024**] Network **106** facilitates wireless or wireline communication between computer server **102** and any other computer, such as clients **120**. Indeed, while illustrated as residing between server **102** and client **120**, network **106** may also reside between various nodes **115** without departing from the scope of the disclosure. In other words, network **106** encompasses any network, networks, or sub-network operable to facilitate communications between various computing components. Network **106** may communicate, for example, Internet Protocol (IP) packets, Frame Relay frames, Asynchronous Transfer Mode (ATM) cells, voice, video, data, and other suitable information between network addresses. Network **106** may include one or more local area networks (LANs), radio access networks (RANs), metropolitan area networks (MANs), wide area networks (WANs) , all or a portion of the global computer network known as the Internet, and/or any other communication system or systems at one or more locations.

[**0025**] In general, disk farm **140** is any memory, database or storage area network (SAN) for storing jobs **150**, profiles, boot images, or other HPC information. According to the illustrated embodiment, disk farm **140** includes one or more storage clients **142**. Disk farm **140** may process and route data packets according to any of a number of communication protocols, for example, InfiniBand (IB), Gigabit Ethernet (GE), or FibreChannel (FC). Data packets are typically used to transport data within disk farm **140**. A data packet may include a header that has a source identifier and a destination identifier. The source identifier, for example, a source address, identifies the transmitter of information, and the destination identifier, for example, a destination address, identifies the recipient of the information.

[**0026**] Client **120** is any device operable to present the user with a job submission screen or administration via a graphical user interface (GUI) **126**. At a high level, illustrated client **120** includes at least GUI **126** and comprises an electronic computing device operable to receive, transmit, process and store any appropriate data associated with system **100**. It will be understood that there may be any number of clients **120** communicably coupled to server **102**. Further, "client **120**" and "user of client **120**" may be used interchangeably as appropriate without departing from the scope of this disclosure. Moreover, for ease of illustration, each client is described in terms of being used by one user. But this disclosure contemplates that many users may use one computer to communicate jobs **150** using the same GUI **126**.

[**0027**] As used in this disclosure, client **120** is intended to encompass a personal computer, touch screen terminal, workstation, network computer, kiosk, wireless data port, cell phone, personal data assistant (PDA) , one or more processors within these or other devices, or any other suitable processing device. For example, client **120** may comprise a computer that includes an input device, such as a keypad, touch screen, mouse, or other device that can accept information, and an output device that conveys information associated with the operation of server **102** or clients **120**, including digital data, visual information, or GUI **126**. Both the input device and output device may include fixed or removable storage media such as a magnetic computer disk, CD-ROM, or other suitable media to both receive input from and provide output to users of clients **120** through the administration and job submission display, namely GUI **126**.

[**0028**] GUI **126** comprises a graphical user interface operable to allow i) the user of client **120** to interface with system **100** to submit one or more jobs **150**; and/or ii) the system (or network) administrator using client **120** to interface with system **100** for any suitable supervisory purpose. Generally, GUI **126** provides the user of client **120** with an efficient and user-friendly presentation of data provided by HPC system **100**. GUI **126** may comprise a plurality of customizable frames or views having interactive fields, pull-down lists, and buttons operated by the user. In one embodiment, GUI **126** presents a job submission display that presents the various job parameter fields and receives commands from the user of client **120** via one of the input devices. GUI **126** may, alternatively or in combination, present the physical and logical status of nodes **115** to the system administrator, as illustrated in FIGS. **4**A-B, and receive various commands from the administrator. Administrator commands may include marking nodes as (un)available, shutting down nodes for maintenance, rebooting nodes, or any other suitable command. Moreover, it should be understood that the term graphical user interface may be used in the singular or in the plural to describe one or more graphical user interfaces and each of the displays of a particular graphical user interface. Therefore, GUI **126** contemplates any graphical user interface, such as a generic web browser, that processes information in system **100** and efficiently presents the results to the user. Server **102** can accept data from client **120** via the web browser (e.g., Microsoft Internet Explorer or Netscape Navigator) and return the appropriate HTML or XML responses using network **106**.

[**0029**] In one aspect of operation, HPC server **102** is first initialized or booted. During this process, cluster management engine **130** determines the existence, state, location, and/or other characteristics of nodes **115** in grid **110**. As described above, this may be based on a "heartbeat" communicated upon each node's initialization or upon near immediate polling by management node **105**. Next, cluster management engine **130** may dynamically allocate various portions of grid **110** to one or more virtual clusters **220** based on, for example, predetermined policies. In one embodiment, cluster management engine **130** continuously moni-

tors nodes **115** for possible failure and, upon determining that one of the nodes **115** failed, effectively managing the failure using any of a variety of recovery techniques. Cluster management engine **130** may also manage and provide a unique execution environment for each allocated node of virtual cluster **220**. The execution environment may consist of the hostname, IP address, operating system, configured services, local and shared file systems, and a set of installed applications and data. The cluster management engine **130** may dynamically add or subtract nodes from virtual cluster **220** according to associated policies and according to inter-cluster policies, such as priority.

[**0030**] When a user logs on to client **120**, he may be presented with a job submission screen via GUI **126**. Once the user has entered the job parameters and submitted job **150**, cluster management engine **130** processes the job submission, the related parameters, and any predetermined policies associated with job **150**, the user, or the user group. Cluster management engine **130** then determines the appropriate virtual cluster **220** based, at least in part, on this information. Engine **130** then dynamically allocates a job space **230** within virtual cluster **220** and executes job **150** across the allocated nodes **115** using HPC techniques. Based, at least in part, on the increased I/O performance, HPC server **102** may more quickly complete processing of job **150**. Upon completion, cluster management engine communicates results **160** to the user.

[**0031**] FIGS. **2**A-D illustrate various embodiments of grid **210** in system **100** and the usage or topology thereof. **FIG. 2A** illustrates one configuration, namely a 3D Torus, of grid **210** using a plurality of node types. For example, the illustrated node types are external I/O node, FS server, FS metadata server, database server, and compute node. **FIG. 2B** illustrates an example of "folding" of grid **210**. Folding generally allows for one physical edge of grid **215** to connect to a corresponding axial edge, thereby providing a more robust or edgeless topology. In this embodiment, nodes **215** are wrapped around to provide a near seamless topology connect by node link **216**. Node line **216** may be any suitable hardware implementing any communications protocol for interconnecting two or more nodes **215**. For example, node line **216** may be copper wire or fiber optic cable implementing Gigabit Ethernet.

[**0032**] **FIG. 2C** illustrates grid **210** with one virtual cluster **220** allocated within it. While illustrated with only one virtual cluster **220**, there may be any number (including zero) of virtual clusters **220** in grid **210** without departing from the scope of this disclosure. Virtual cluster **220** is a logical grouping of nodes **215** for processing related jobs **150**. For example, virtual cluster **220** may be associated with one research group, a department, a lab, or any other group of users likely to submit similar jobs **150**. Virtual cluster **220** may be any shape and include any number of nodes **215** within grid **210**. Indeed, while illustrated virtual cluster **220** includes a plurality of physically neighboring nodes **215**, cluster **220** may be a distributed cluster of logically related nodes **215** operable to process job **150**.

[**0033**] Virtual cluster **220** may be allocated at any appropriate time. For example, cluster **220** may be allocated upon initialization of system **100** based, for example, on startup parameters or may be dynamically allocated based, for example, on changed server **102** needs. Moreover, virtual cluster **220** may change its shape and size over time to quickly respond to changing requests, demands, and situations. For example, virtual cluster **220** may be dynamically changed to include an automatically allocated first node **215** in response to a failure of a second node **215**, previously part of cluster **220**. In certain embodiments, clusters **220** may share nodes **215** as processing requires.

[**0034**] **FIG. 2D** illustrates various job spaces, **230***a* and **230***b* respectively, allocated within example virtual cluster **220**. Generally, job space **230** is a set of nodes **215** within virtual cluster **220** dynamically allocated to complete received job **150**. Typically, there is one job space **230** per executing job **150** and vice versa, but job spaces **230** may share nodes **215** without departing from the scope of the disclosure. The dimensions of job space **230** may be manually input by the user or administrator or dynamically determined based on job parameters, policies, and/or any other suitable characteristic.

[**0035**] FIGS. **3**A-C illustrate various embodiments of individual nodes **115** in grid **110**. In the illustrated, but example, embodiments, nodes **115** are represented by blades **315**. Blade **315** comprises any computing device in any orientation operable to process all or a portion, such as a thread or process, of job **150**. For example, blade **315** may be a standard Xeon64™ motherboard, a standard PCI-Express Opteron™ motherboard, or any other suitable computing card.

[**0036**] Blade **315** is an integrated fabric architecture that distributes the fabric switching components uniformly across nodes **115** in grid **110**, thereby possibly reducing or eliminating any centralized switching function, increasing the fault tolerance, and allowing message passing in parallel. More specifically, blade **315** includes an integrated switch **345**. Switch **345** includes any number of ports that may allow for different topologies. For example, switch **345** may be an eight-port switch that enables a tighter three-dimensional mesh or 3D Torus topology. These eight ports include two "X" connections for linking to neighbor nodes **115** along an X-axis, two "Y" connections for linking to neighbor nodes **115** along a Y-axis, two "Z" connections for linking to neighbor nodes **115** along a Z-axis, and two connections for linking to management node **105**. In one embodiment, switch **345** may be a standard eight port Infiniband-4x switch IC, thereby easily providing built-in fabric switching. Switch **345** may also comprise a twenty-four port switch that allows for multidimensional topologies, such a 4-D Torus, or other non-traditional topologies of greater than three dimensions. Moreover, nodes **115** may further interconnected along a diagonal axis, thereby reducing jumps or hops of communications between relatively distant nodes **115**. For example, a first node **115** may be connected with a second node **115** that physically resides along a northeasterly axis several three dimensional "jumps" away.

[**0037**] **FIG. 3A** illustrates a blade **315** that, at a high level, includes at least two processors **320***a* and **320***b*, local or remote memory **340**, and integrated switch (or fabric) **345**. Processor **320** executes instructions and manipulates data to perform the operations of blade **315** such as, for example, a central processing unit (CPU). Reference to processor **320** is meant to include multiple processors **320** where applicable. In one embodiment, processor **320** may comprise a Xeon64

or Itanium™ processor or other similar processor or derivative thereof. For example, the Xeon64 processor may be a 3.4 GHz chip with a 2 MB Cache and HyperTreading. In this embodiment, the dual processor module may include a native PCI/Express that improves efficiency. Accordingly, processor **320** has efficient memory bandwidth and, typically, has the memory controller built into the processor chip.

[**0038**] Blade **315** may also include Northbridge **321**, Southbridge **322**, PCI channel **325**, HCA **335**, and memory **340**. Northbridge **321** communicates with processor **320** and controls communications with memory **340**, a PCI bus, Level 2 cache, and any other related components. In one embodiment, Northbridge **321** communicates with processor **320** using the frontside bus (FSB). Southbridge **322** manages many of the input/output (I/O) functions of blade **315**. In another embodiment, blade **315** may implement the Intel Hub Architecture (IHATM), which includes a Graphics and AGP Memory Controller Hub (GMCH) and an I/O Controller Hub (ICH).

[**0039**] PCI channel **325** comprises any high-speed, low latency link designed to increase the communication speed between integrated components. This helps reduce the number of buses in blade **315**, which can reduce system bottlenecks. HCA **335** comprises any component providing channel-based I/O within server **102**. Each HCA **335** may provide a total bandwidth of 2.65 GB/sec, thereby allowing 1.85 GB/sec per PE to switch **345** and 800 MB/sec per PE to I/O such as, for example, BIOS (Basic Input/Output System), an Ethernet management interface, and others. This further allows the total switch **345** bandwidth to be 3.7 GB/sec for 13.6 Gigaflops/sec peak or 0.27 Bytes/Flop I/O rate is 50 MB/sec per Gigaflop.

[**0040**] Memory **340** includes any memory or database module and may take the form of volatile or non-volatile memory including, without limitation, magnetic media, optical media, flash memory, random access memory (RAM), read-only memory (ROM), removable media, or any other suitable local or remote memory component. In the illustrated embodiment, memory **340** is comprised of 8 GB of dual double data rate (DDR) memory components operating at least 6.4 GB/s. Memory **340** may include any appropriate data for managing or executing HPC jobs **150** without departing from this disclosure.

[**0041**] **FIG. 3B** illustrates a blade **315** that includes two processors **320***a* and **320***b*, memory **340**, HyperTransport/peripheral component interconnect (HT/PCI) bridges **330***a* and **330***b*, and two HCAs **335***a* and **335***b*.

[**0042**] Example blade **315** includes at least two processors **320**. Processor **320** executes instructions and manipulates data to perform the-operations of blade **315** such as, for example, a central processing unit (CPU). In the illustrated embodiment, processor **320** may comprise an Opteron processor or other similar processor or derivative. In this embodiment, the Opteron processor design supports the development of a well balanced building block for grid **110**. Regardless, the dual processor module may provide four to five Gigaflop usable performance and the next generation technology helps solve memory bandwidth limitation. But blade **315** may more than two processors **320** without departing from the scope of this disclosure. Accordingly, processor **320** has efficient memory bandwidth and, typically, has the memory controller built into the processor chip. In this embodiment, each processor **320** has one or more HyperTransport™ (or other similar conduit type) links **325**.

[**0043**] Generally, HT link **325** comprises any high-speed, low latency link designed to increase the communication speed between integrated components. This helps reduce the number of buses in blade **315**, which can reduce system bottlenecks. HT link **325** supports processor to processor communications for cache coherent multiprocessor blades **315**. Using HT links **325**, up to eight processors **320** may be placed on blade **315**. If utilized, HyperTransport may provide bandwidth of 6.4 GB/sec, 12.8, or more, thereby providing a better than forty-fold increase in data throughput over legacy PCI buses. Further HyperTransport technology may be compatible with legacy I/O standards, such as PCI, and other technologies, such as PCI-X.

[**0044**] Blade **315** further includes HT/PCI bridge **330** and HCA **335**. PCI bridge **330** may be designed in compliance with PCI Local Bus Specification Revision 2.2 or 3.0 or PCI Express Base Specification 1.0a or any derivatives thereof. HCA **335** comprises any component providing channel-based I/O within server **102**. In one embodiment, HCA **335** comprises an Infiniband HCA. InfiniBand channels are typically created by attaching host channel adapters and target channel adapters, which enable remote storage and network connectivity into an InfiniBand fabric, illustrated in more detail in **FIG. 3B**. Hypertransport **325** to PCI-Express Bridge **330** and HCA **335** may create a full-duplex 2 GB/sec I/O channel for each processor **320**. In certain embodiments, this provides sufficient bandwidth to support processor-processor communications in distributed HPC environment **100**. Further, this provides blade **315** with I/O performance nearly or substantially balanced with the performance of processors **320**.

[**0045**] **FIG. 3C** illustrates another embodiment of blade **315** including a daughter board. In this embodiment, the daughter board may support 3.2 GB/sec or higher cache coherent interfaces. The daughter board is operable to include one or more Field Programmable Gate Arrays (FPGAs) **350**. For example, the illustrated daughter board includes two FPGAs **350**, represented by **350***a* and **350***b*, respectively. Generally, FPGA **350** provides blade **315** with non-standard interfaces, the ability to process custom algorithms, vector processors for signal, image, or encryption/decryption processing applications, and high bandwidth. For example, FPGA may supplement the ability of blade **315** by providing acceleration factors of ten to twenty times the performance of a general purpose processor for special functions such as, for example, low precision Fast Fourier Transform (FFT) and matrix arithmetic functions.

[**0046**] The preceding illustrations and accompanying descriptions provide exemplary diagrams for implementing various scaleable nodes **115** (illustrated as example blades **315**). However, these figures are merely illustrative and system **100** contemplates using any suitable combination and arrangement of elements for implementing various scalability schemes. Although the present invention has been illustrated and described, in part, in regard to blade server **102**, those of ordinary skill in the art will recognize that the teachings of the present invention may be applied to any clustered HPC server environment. Accordingly, such clus-

tered servers **102** that incorporate the techniques described herein may be local or a distributed without departing from the scope of this disclosure. Thus, these servers **102** may include HPC modules (or nodes **115**) incorporating any suitable combination and arrangement of elements for providing high performance computing power, while reducing I/O latency. Moreover, the operations of the various illustrated HPC modules may be combined and/or separated as appropriate. For example, grid **110** may include a plurality of substantially similar nodes **115** or various nodes **115** implementing differing hardware or fabric architecture.

[**0047**] FIGS. **4**A-B illustrate various embodiments of a management graphical user interface **400** in accordance with the system **100**. Often, management GUI **400** is presented to client **120** using GUI **126**. In general, management GUI **400** presents a variety of management interactive screens or displays to a system administrator and/or a variety of job submission or profile screens to a user. These screens or displays are comprised of graphical elements assembled into various views of collected information. For example, GUI **400** may present a display of the physical health of grid **110** (illustrated in **FIG. 4A**) or the logical allocation or topology of nodes **115** in grid **110** (illustrated in **FIG. 4B**).

[**0048**] **FIG. 4A** illustrates example display **400***a*. Display **400***a* may include information presented to the administrator for effectively managing nodes **115**. The illustrated embodiment includes a standard web browser with a logical "picture" or screenshot of grid **110**. For example, this picture may provide the physical status of grid **110** and the component nodes **115**. Each node **115** may be one of any number of colors, with each color representing various states. For example, a failed node **115** may be red, a utilized or allocated node **115** may be black, and an unallocated node **115** may be shaded. Further, display **400***a* may allow the administrator to move the pointer over one of the nodes **115** and view the various physical attributes of it. For example, the administrator may be presented with information including "node,""availability,""processor utilization,""memory utilization,""temperature,""physical location," and "address." Of course, these are merely example data fields and any appropriate physical or logical node information may be display for the administrator. Display **400***a* may also allow the administrator to rotate the view of grid **110** or perform any other suitable function.

[**0049**] **FIG. 4B** illustrates example display **400***b*. Display **400***b* presents a view or picture of the logical state of grid **100**. The illustrated embodiment presents the virtual cluster **220** allocated within grid **110**. Display **400***b* further displays two example job spaces **230** allocate within cluster **220** for executing one or more jobs **150**. Display **400***b* may allow the administrator to move the pointer over graphical virtual cluster **220** to view the number of nodes **115** grouped by various statuses (such as allocated or unallocated) . Further, the administrator may move the pointer over one of the job spaces **230** such that suitable job information is presented. For example, the administrator may be able to view the job name, start time, number of nodes, estimated end time, processor usage, I/O usage, and others.

[**0050**] It will be understood that management GUI **126** (represented above by example displays **400***a* and **400***b*, respectively) is for illustration purposes only and may include none, some, or all of the illustrated graphical elements as well as additional management elements not shown.

[**0051**] **FIG. 5** illustrates one embodiment of cluster management engine **130**, shown here as engine **500**, in accordance with system **100**. In this embodiment, cluster management engine **500** includes a plurality of sub-modules or components: physical manager **505**, virtual manager **510**, job scheduler **515**, and local memory or variables **520**.

[**0052**] Physical manager **505** is any software, logic, firmware, or other module operable to determine the physical health of various nodes **115** and effectively manage nodes **115** based on this determined health. Physical manager may use this data to efficiently determine and respond to node **115** failures. In one embodiment, physical manager **505** is communicably coupled to a plurality of agents **132**, each residing on one node **115**. As described above, agents **132** gather and communicate at least physical information to manager **505**. Physical manager **505** may be further operable to communicate alerts to a system administrator at client **120** via network **106**.

[**0053**] Virtual manager **510** is any software, logic, firmware, or other module operable to manage virtual clusters **220** and the logical state of nodes **115**. Generally, virtual manager **510** links a logical representation of node **115** with the physical status of node **115**. Based on these links, virtual manager **510** may generate virtual clusters **220** and process various changes to these clusters **220**, such as in response to node failure or a (system or user) request for increased HPC processing. Virtual manager **510** may also communicate the status of virtual cluster **220**, such as unallocated nodes **115**, to job scheduler **515** to enable dynamic backfilling of unexecuted, or queued, HPC processes and jobs **150**. Virtual manager **510** may further determine the compatibility of job **150** with particular nodes **115** and communicate this information to job scheduler **515**. In certain embodiments, virtual manager **510** may be an object representing an individual virtual cluster **220**.

[**0054**] Cluster management engine **500** may also include job scheduler **515**. Job scheduler sub-module **515** is a topology-aware module that processes aspects of the system's resources, as well with the processors and the time allocations, to determine an optimum job space **230** and time. Factors that are often considered include processors, processes, memory, interconnects, disks, visualization engines, and others. In other words, job scheduler **515** typically interacts with GUI **126** to receive jobs **150**, physical manager **505** to ensure the health of various nodes **115**, and virtual manager **510** to dynamically allocate job space **230** within a certain virtual cluster **220**. This dynamic allocation is accomplished through various algorithms that often incorporates knowledge of the current topology of grid **110** and, when appropriate, virtual cluster **220**. Job scheduler **515** handles both batch and interactive execution of both serial and parallel programs. Scheduler **515** should also provide a way to implement policies **524** on selecting and executing various problems presented by job **150**.

[**0055**] Cluster management engine **500**, such as through job scheduler **515**, may be further operable to perform efficient check-pointing. Restart dumps typically comprise over seventy-five percent of data written to disk. This I/O is often done so that processing is not lost to a platform failure.

Based on this, a file system's I/O can be segregated into two portions: productive I/O and defensive I/O. Productive I/O is the writing of data that the user calls for to do science such as, for example, visualization dumps, traces of key physics variables over time, and others. Defensive I/O is performed to manage a large simulation run over a substantial period of time. Accordingly, increased I/O bandwidth greatly reduces the time and risk involved in check-pointing.

[**0056**] Returning to engine **500**, local memory **520** comprises logical descriptions (or data structures) of a plurality of features of system **100**. Local memory **520** may be stored in any physical or logical data storage operable to be defined, processed, or retrieved by compatible code. For example, local memory **520** may comprise one or more extensible Markup Language (XML) tables or documents. The various elements may be described in terms of SQL statements or scripts, Virtual Storage Access Method (VSAM) files, flat files, binary data files, Btrieve files, database files, or comma-separated-value (CSV) files. It will be understood that each element may comprise a variable, table, or any other suitable data structure. Local memory **520** may also comprise a plurality of tables or files stored on one server **102** or across a plurality of servers or nodes. Moreover, while illustrated as residing inside engine **500**, some or all of local memory **520** may be internal or external without departing from the scope of this disclosure.

[**0057**] Illustrated local memory **520** includes physical list **521**, virtual list **522**, group file **523**, policy table **524**, and job queue **525**. But, while not illustrated, local memory **520** may include other data structures, including a job table and audit log, without departing from the scope of this disclosure. Returning to the illustrated structures, physical list **521** is operable to store identifying and physical management information about node **115**. Physical list **521** may be a multi-dimensional data structure that includes at least one record per node **115**. For example, the physical record may include fields such as "node,""availability,""processor utilization,""memory utilization,""temperature,""physical location,""address,""boot images," and others. It will be understood that each record may include none, some, or all of the example fields. In one embodiment, the physical record may provide a foreign key to another table, such as, for example, virtual list **522**.

[**0058**] Virtual list **522** is operable to store logical or virtual management information about node **115**. Virtual list **522** may be a multi-dimensional data structure that includes at least one record per node **115**. For example, the virtual record may include fields such as "node,""availability,""job, ""virtual cluster,""secondary node,""logical location, ""compatibility," and others. It will be understood that each record may include none, some, or all of the example fields. In one embodiment, the virtual record may include a link to another table such as, for example, group file **523**.

[**0059**] Group file **523** comprises one or more tables or records operable to store user group and security information, such as access control lists (or ACLs). For example, each group record may include a list of available services, nodes **115**, or jobs for a user. Each logical group may be associated with a business group or unit, a department, a project, a security group, or any other collection of one or more users that are able to submit jobs **150** or administer at least part of system **100**. Based on this information, cluster management engine **500** may determine if the user submitting job **150** is a valid user and, if so, the optimum parameters for job execution. Further, group table **523** may associate each user group with a virtual cluster **220** or with one or more physical nodes **115**, such as nodes residing within a particular group's domain. This allows each group to have an individual processing space without competing for resources. However, as described above, the shape and size of virtual cluster **220** may be dynamic and may change according to needs, time, or any other parameter.

[**0060**] Policy table **524** includes one or more policies. It will be understood that policy table **524** and policy **524** may be used interchangeably as appropriate. Policy **524** generally stores processing and management information about jobs **150** and/or virtual clusters **220**. For example, policies **524** may include any number of parameters or variables including problem size, problem run time, timeslots, preemption, users' allocated share of node **115** or virtual cluster **220**, and such.

[**0061**] Job queue **525** represents one or more streams of jobs **150** awaiting execution. Generally, queue **525** comprises any suitable data structure, such as a bubble array, database table, or pointer array, for storing any number (including zero) of jobs **150** or reference thereto. There may be one queue **525** associated with grid **110** or a plurality of queues **525**, with each queue **525** associated with one of the unique virtual clusters **220** within grid **110**.

[**0062**] In one aspect of operation, cluster management engine **500** receives job **150**, made up of N tasks which cooperatively solve a problem by performing calculations and exchanging information. Cluster management engine **500** allocates N nodes **115** and assigns each of the N tasks to one particular node **515** using any suitable technique, thereby allowing the problem to be solved efficiently. For example, cluster management engine **500** may utilize job parameters, such as job task placement strategy, supplied by the user. Regardless, cluster management engine **500** attempts to exploit the architecture of server **102**, which in turn provides the quicker turnaround for the user and likely improves the overall throughput for system **100**.

[**0063**] In one embodiment, cluster management engine **500** then selects and allocates nodes **115** according to any of the following example topologies:

[**0064**] Specified 2D (x,y) or 3D (x,y,z)—Nodes **115** are allocated and tasks may be ordered in the specified dimensions, thereby preserving efficient neighbor to neighbor communication. The specified topology manages a variety of jobs **150** where it is desirable that the physical communication topology match the problem topology allowing the cooperating tasks of job **150** to communicate frequently with neighbor tasks. For example, a request of 8 tasks in a 2×2×2 dimension (2, 2, 2) will be allocated in a cube. For best-fit purposes, 2D allocations can be "folded" into 3 dimensions (as discussed in **FIG. 2D**), while preserving efficient neighbor to neighbor communications. Cluster management engine **500** may be free to allocate the specified dimensional shape in any orientation. For example, a 2×2×8 box may be allocated within the available physical nodes vertically or horizontally

[**0065**] Best Fit Cube—cluster management engine **500** allocates N nodes **115** in a cubic volume. This topology

efficiently handles jobs **150** allowing cooperating tasks to exchange data with any other tasks by minimizing the distance between any two nodes **115**.

[**0066**] Best Fit Sphere—cluster management engine **500** allocates N nodes **115** in a spherical volume. For example, the first task may be placed in the center node **115** of the sphere with the rest of the tasks placed on nodes **115** surrounding the center node **115**. It will be understood that the placement order of the remaining tasks is not typically critical. This topology may minimize the distance between the first task and all other tasks. This efficiently handles a large class of problems where tasks **2**-N communicate with the first task, but not with each other.

[**0067**] Random—cluster management engine **500** allocates N nodes **115** with reduced consideration for where nodes **115** are logically or physically located. In one embodiment, this topology encourages aggressive use of grid **110** for backfilling purposes, with little impact to other jobs **150**.

[**0068**] It will be understood that the prior topologies and accompanying description are for illustration purposes only and may not depict actual topologies used or techniques for allocating such topologies.

[**0069**] Cluster management engine **500** may utilize a placement weight, stored as a job **150** parameter or policy **524** parameter. In one embodiment, the placement weight is a modifier value between 0 and 1, which represents how aggressively cluster management engine **500** should attempt to place nodes **115** according to the requested task (or process) placement strategy. In this example, a value of 0 represents placing nodes **115** only if the optimum strategy (or dimensions) is possible and a value of 1 represents placing nodes **115** immediately, as long as there are enough free or otherwise available nodes **115** to handle the request. Typically, the placement weight does not override administrative policies **524** such as resource reservation, in order to prevent starvation of large jobs **150** and preserve the job throughput of HPC system **100**.

[**0070**] The preceding illustration and accompanying description provide an exemplary modular diagram for engine **500** implementing logical schemes for managing nodes **115** and jobs **150**. However, this figure is merely illustrative and system **100** contemplates using any suitable combination and arrangement of logical elements for implementing these and other algorithms. Thus, these software modules may include any suitable combination and arrangement of elements for effectively managing nodes **115** and jobs **150**. Moreover, the operations of the various illustrated modules may be combined and/or separated as appropriate.

[**0071**] **FIG. 6** is a flowchart illustrating an example method **600** for dynamically processing a job submission in accordance with one embodiment of the present disclosure. Generally, **FIG. 6** describes method **600**, which receives a batch job submission, dynamically allocates nodes **115** into a job space **230** based on the job parameters and associated policies **524**, and executes job **150** using the allocated space. The following description focuses on the operation of cluster management module **130** in performing method **600**. But system **100** contemplates using any appropriate combination and arrangement of logical elements implementing some or all of the described functionality, so long as the functionality remains appropriate.

[**0072**] Method **600** begins at step **605**, where HPC server **102** receives job submission **150** from a user. As described above, in one embodiment the user may submit job **150** using client **120**. In another embodiment, the user may submit job **150** directly using HPC server **102**. Next, at step **610**, cluster management engine **130** selects group **523** based upon the user. Once the user is verified, cluster management engine **130** compares the user to the group access control list (ACL) at step **615**. But it will be understood that cluster management engine **130** may use any appropriate security technique to verify the user. Based upon determined group **523**, cluster management engine **130** determines if the user has access to the requested service. Based on the requested service and hostname, cluster management engine **130** selects virtual cluster **220** at step **620**. Typically., virtual cluster **220** may be identified and allocated prior to the submission of job **150**. But, in the event virtual cluster **220** has not been established, cluster management engine **130** may automatically allocate virtual cluster **220** using any of the techniques described above. Next, at step **625**, cluster management engine **130** retrieves policy **524** based on the submission of job **150**. In one embodiment, cluster management engine **130** may determine the appropriate policy **524** associated with the user, job **150**, or any other appropriate criteria. Cluster management engine **130** then determines or otherwise calculates the dimensions of job **150** at step **630**. It will be understood that the appropriate dimensions may include length, width, height, or any other appropriate parameter or characteristic. As described above, these dimensions are used to determine the appropriate job space **230** (or subset of nodes **115**) within virtual cluster **220**. After the initial parameters have been established, cluster management **130** attempts to execute job **150** on HPC server **102** in steps **635** through **665**.

[**0073**] At decisional step **635**, cluster management engine **130** determines if there are enough available nodes to allocate the desired job space **230**, using the parameters already established. If there are not enough nodes **115**, then cluster management engine **130** determines the earliest available subset **230** of nodes **115** in virtual cluster **220** at step **640**. Then, cluster management engine **130** adds job **150** to job queue **125** until the subset **230** is available at step **645**. Processing then returns to decisional step **635**. Once there are enough nodes **115** available, then cluster management engine **130** dynamically determines the optimum subset **230** from available nodes **115** at step **650**. It will be understood that the optimum subset **230** may be determined using any appropriate criteria, including fastest processing time, most reliable nodes **115**, physical or virtual locations, or first available nodes **115**. At step **655**, cluster management engine **130** selects the determined subset **230** from the selected virtual cluster **220**. Next, at step **660**, cluster management engine **130** allocates the selected nodes **115** for job **150** using the selected subset **230**. According to one embodiment, cluster management engine **130** may change the status of nodes **115** in virtual node list **522** from "unallocated" to "allocated". Once subset **230** has been appropriately allocated, cluster management engine **130** executes job **150** at step **665** using the allocated space based on the job parameters, retrieved policy **524**, and any other suitable parameters. At any appropriate time, cluster management engine **130** may communicate or otherwise present job results **160** to the user. For example, results **160** may be formatted and presented to the user via GUI **126**.

[0074] **FIG. 7** is a flowchart illustrating an example method **700** for dynamically backfilling a virtual cluster **220** in grid **110** in accordance with one embodiment of the present disclosure. At a high level, method **700** describes determining available space in virtual cluster **220**, determining the optimum job **150** that is compatible with the space, and executing the determined job **150** in the available space. The following description will focus on the operation of cluster management module **130** in performing this method. But, as with the previous flowchart, system **100** contemplates using any appropriate combination and arrangement of logical elements implementing some or all of the described functionality.

[0075] Method **700** begins at step **705**, where cluster management engine **130** sorts job queue **525**. In the illustrated embodiment, cluster management engine **130** sorts the queue **525** based on the priority of jobs **150** stored in the queue **525**. But it will be understood that cluster management engine **130** may sort queue **525** using any suitable characteristic such that the appropriate or optimal job **150** will be executed. Next, at step **710**, cluster management engine **130** determines the number of available nodes **115** in one of the virtual clusters **220**. Of course, cluster management engine **130** may also determine the number of available nodes **115** in grid **110** or in any one or more of virtual clusters **220**. At step **715**, cluster management engine **130** selects first job **150** from sorted job queue **525**. Next, cluster management engine **130** dynamically determines the optimum shape (or other dimensions) of selected job **150** at **720**. Once the optimum shape or dimension of selected job **150** is determined, then cluster management engine **130** determines if it can backfill job **150** in the appropriate virtual cluster **220** in steps **725** through **745**.

[0076] At decisional step **725**, cluster management engine **130** determines if there are enough nodes **115** available for the selected job **150**. If there are enough available nodes **115**, then at step **730** cluster management engine **130** dynamically allocates nodes **115** for the selected job **150** using any appropriate technique. For example, cluster management engine **130** may use the techniques describes in **FIG. 6**. Next, at step **735**, cluster management engine **130** recalculates the number of available nodes in virtual cluster **220**. At step **740**, cluster management engine **130** executes job **150** on allocated nodes **115**. Once job **150** has been executed (or if there were not enough nodes **115** for selected job **150**), then cluster management engine **130** selects the next job **150** in the sorted job queue **525** at step **745** and processing returns to step **720**. It will be understood that while illustrated as a loop, cluster management engine **130** may initiate, execute, and terminate the techniques illustrated in method **700** at any appropriate time.

[0077] **FIG. 8** is a flowchart illustrating an example method **800** for dynamically managing failure of a node **115** in grid **110** in accordance with one embodiment of the present disclosure. At a high level, method **800** describes determining that node **115** failed, automatically performing job recovery and management, and replacing the failed node **115** with a secondary node **115**. The following description will focus on the operation of cluster management module **130** in performing this method. But, as with the previous flowcharts, system **100** contemplates using any appropriate combination and arrangement of logical elements implementing some or all of the described functionality.

[0078] Method **800** begins at step **805**, where cluster management engine **130** determines that node **115** has failed. As described above, cluster management engine **130** may determine that node **115** has failed using any suitable technique. For example, cluster management engine **130** may pull nodes **115** (or agents **132**) at various times and may determine that node **115** has failed based upon the lack of a response from node **115**. In another example, agent **132** existing on node **115** may communicate a "heartbeat" and the lack of this "heartbeat" may indicate node **115** failure. Next, at step **810**, cluster management engine **130** removes the failed node **115** from virtual cluster **220**. In one embodiment, cluster management engine **130** may change the status of node **115** in virtual list **522** from "allocated" to "failed". Cluster management engine **130** then determines if a job **150** is associated with failed node **115** at decisional step **815**. If there is no job **150** associated with node **115**, then processing ends. As described above, before processing ends, cluster management engine **130** may communicate an error message to an administrator, automatically determine a replacement node **115**, or any other suitable processing. If there is a job **150** associated with the failed node **115**, then the cluster management engine **130** determines other nodes **115** associated with the job **150** at step **820**. Next, at step **825**, cluster management engine **130** kills job **150** on all appropriate nodes **115**. For example, cluster management engine **130** may execute a kill job command or use any other appropriate technique to end job **150**. Next, at step **830**, cluster management engine **130** de-allocates nodes **115** using virtual list **522**. For example, cluster management engine **130** may change the status of nodes **115** in virtual list **522** from "allocated" to "available". Once the job has been terminated and all appropriate nodes **115** de-allocated, then cluster management engine **130** attempts to re-execute the job **150** using available nodes **115** in steps **835** through **850**.

[0079] At step **835**, cluster management engine **130** retrieves policy **524** and parameters for the killed job **150** at step **835**. Cluster management engine **130** then determines the optimum subset **230** of nodes **115** in virtual cluster **220**, at step **840**, based on the retrieved policy **524** and the job parameters. Once the subset **230** of nodes **115** has been determined, then cluster management engine **130** dynamically allocates the subset **230** of nodes **115** at step **845**. For example, cluster management engine **130** may change the status of nodes **115** in virtual list **522** from "unallocated" to "allocated". It will be understood that this subset of nodes **115** may be different from the original subset of nodes that job **150** was executing on. For example, cluster management engine **130** may determine that a different subset of nodes is optimal because of the node failure that prompted this execution. In another example, cluster management engine **130** may have determined that a secondary node **115** was operable to replace the failed node **115** and the new subset **230** is substantially similar to the old job space **230**. Once the allocated subset **230** has been determined and allocated, then cluster management engine **130** executes job **150** at step **850**.

[0080] The preceding flowcharts and accompanying description illustrate exemplary methods **600**, **700**, and **800**. In short, system **100** contemplates using any suitable technique for performing these and other tasks. Accordingly, many of the steps in this flowchart may take place simultaneously and/or in different orders than as shown. More-

over, system **100** may use methods with additional steps, fewer steps, and/or different steps, so long as the methods remain appropriate.

[**0081**] Although this disclosure has been described in terms of certain embodiments and generally associated methods, alterations and permutations of these embodiments and methods will be apparent to those skilled in the art. Accordingly, the above description of example embodiments does not define or constrain this disclosure. Other changes, substitutions, and alterations are also possible without departing from the spirit and scope of this disclosure.

What is claimed is:

1. A High Performance Computing (HPC) node comprising:

a motherboard;

a switch comprising eight or more ports, the switch integrated on the motherboard; and

at least two processors operable to execute an HPC job, each processor communicably coupled to the integrated switch and integrated on the motherboard.

2. The HPC node of claim 1, each processor coupled to the integrated switch through a Host Channel Adapter (HCA).

3. The HPC node of claim 2, each processor further coupled to the integrated switch through a Hyper Transport/PCI bridge.

4. The HPC node of claim 1, the processors communicably coupled via a Hyper Transport link.

5. The HPC node of claim 1, each processor communicably coupled to the integrated switch through a North Bridge.

6. The HPC node of claim 1, the integrated switch operable to communicate I/O messages at a bandwidth substantially similar to power of the processors.

7. The HPC node of claim 1, the integrated switch comprising an Infiniband switch.

8. The HPC node of claim 1, the integrated switch operable to:

communicate a first message from a first of the two or more processors; and

communicate a second message from a second of the two or more processors, the first and second message communicated in parallel.

9. A High Performance Computing (HPC) system comprising a plurality of interconnected HPC nodes, each node comprising:

a motherboard;

a switch comprising eight or more ports, the switch integrated on the motherboard and operable to interconnect at least a subset of the plurality of nodes; and

at least two processors operable to execute an HPC job, each processor communicably coupled to the integrated switch and integrated on the motherboard.

10. The HPC system of claim 9, the two or more processors on each node coupled to the integrated switch through a Host Channel Adapter (HCA).

11. The HPC system of claim 10, the two or more processors on each node further coupled to the integrated switch through a Hyper Transport/PCI bridge.

12. The HPC system of claim 9, the two or more processors on each node communicably inter-coupled via a Hyper Transport link.

13. The HPC system of claim 9, the two or more processors on each node communicably coupled to the integrated switch through a North Bridge.

14. The HPC system of claim 9, the integrated switch of each node operable to communicate I/O messages at a bandwidth substantially similar to power of the processors.

15. The HPC system of claim 9, the integrated switch of each node comprising an Infiniband switch.

16. The HPC system of claim 9, the plurality of HPC nodes arranged in a topology, the topology enabled by the integrated fabric of each node.

17. The HPC system of claim 16, the topology comprising a hypercube.

18. The HPC system of claim 16, the topology comprising a folded topology.

19. The HPC system of claim 9, a first node of the plurality of nodes interconnected to a second node of the plurality of nodes along an X axis, a third node of the plurality of nodes along a Y axis, a fourth node of the plurality of nodes along a Z axis, and a fifth node along a diagonal axis.

20. The HPC system of claim 19, the connection between the first node and the fifth node operable to reduce message jumps among the plurality of nodes.

21. A method for forming an HPC node, comprising:

providing a motherboard;

integrating a switch with the motherboard;

integrating at least two processors with the motherboard; and

coupling each processor with the integrated switch.

22. The method of claim 21, wherein coupling each processor with the integrated switch comprises coupling each processor to the integrated switch through a Host Channel Adapter (HCA).

23. The method of claim 22, wherein coupling each processor with the integrated switch comprises coupling each processor to the integrated switch through a Hyper Transport/PCI bridge.

24. The method of claim 21, further comprising coupling the processors via a Hyper Transport link.

25. The method of claim 21, wherein coupling each processor with the integrated switch comprises coupling each processor communicably to the integrated switch through a North Bridge.

26. The method of claim 21, the integrated switch operable to communicate I/O messages at a bandwidth substantially similar to power of the processors.

27. The method of claim 21, the integrated switch comprising an Infiniband switch.

* * * * *