(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
16 February 2006 (16.02.2006)

PCT

(10) International Publication Number
WO 2006/017584 A2

(51) International Patent Classification[7]: G06F 3/06

(21) International Application Number:
PCT/US2005/027587

(22) International Filing Date: 4 August 2005 (04.08.2005)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
10/911,398 4 August 2004 (04.08.2004) US

(71) Applicant *(for all designated States except US)*: **VIRTUAL IRON SOFTWARE, INC.** [US/US]; 900 Chelmsford Street, Tower 1, Floor 2, Lowel, MA 01851 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **VASILEVSKY, Alexander, D.** [US/US]; 5 Gooseneck Lane, Westford, MA 01886 (US). **TRONKOWSKI, Kevin** [US/US]; 109 Noons Quarry Road, Milford,NH 03055 (US). **NOYES, Steven, S.** [US/US]; 294 Upper North Row Road, Sterling, MA 01564 (US).

(74) Agent: **RUSSAVAGE, Edward, J.**; Lowrie, Lando & Anastasi, LLP, One Main Street, Cambridge MA 02142 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

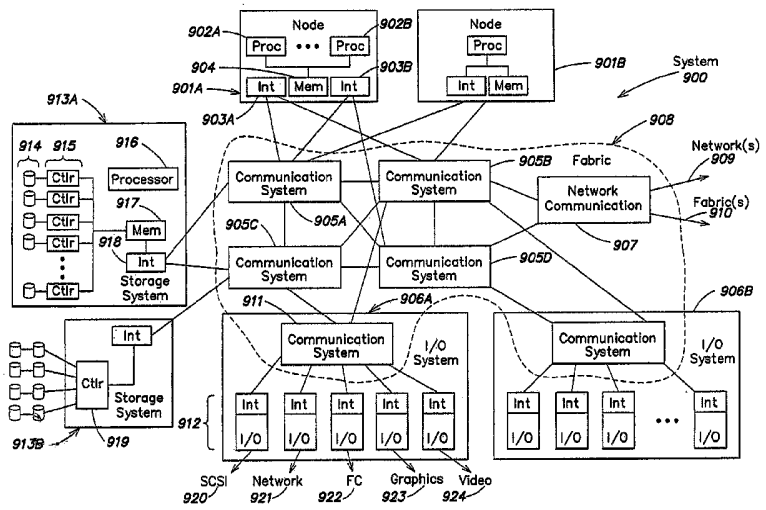
(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: VIRTUAL HOST BUS ADAPTER AND METHOD



(57) Abstract: A virtualized storage adapter architecture and method is provided wherein lower level details of the storage adapter architecture are isolated from an operating system and its applications that execute on a virtualization architecture. This isolation may be performed, for example, by providing a virtual storage adapter that is backed by one or more physical storage adapters. The virtual storage adapter may be referenced by a globally unique identifier. For example, the virtual storage adapter may be referenced by a World Wide Node Name (WWNN). In another example, changes may be made to the underlying physical storage configuration without the need for changes in the virtual storage adapter or its interface to an operating system or its applications.

WO 2006/017584 A2

VIRTUAL HOST BUS ADAPTER AND METHOD

Related Applications

Field of the Invention

The field of the invention relates generally to computer storage, and more particularly, to storage in a virtual computing environment.

Background of the Invention

Conventional datacenters include a complex mesh of N-tier applications. Each tier typically includes multiple servers (nodes) that are dedicated to each application or application portion. These nodes generally include one or more computer systems that execute an application or portion thereof, and provide computing resources to clients. Some systems are general purpose computers (e.g., a Pentium-based server system) having general purpose operating systems (e.g., Microsoft Server 2003) while others are special-purpose systems (e.g., a network attached storage system, database server, etc.) that are specially developed for this particular purpose using custom operating system(s) and hardware. Typically, these servers provide a single function (e.g., file server, application server, backup server, etc.) to one or more client computers coupled through a communication network (e.g., enterprise network, Internet, combination of both).

Configurations of datacenter resources may be adjusted from time to time depending on the changing requirements of the applications used, performance issues, reallocation of resources, and other reasons. Configuration changes are performed, for example, by manually reconfiguring servers, adding memory/storage, etc., and these changes generally involve a reboot of affected computer systems and/or an interruption in the execution of the affected application. There exist other techniques such as server farms with front-end load balancers and grid-aware applications that allow the addition and deletion of resources. Operating systems or applications on which grid-aware applications are supported must be specifically developed to operate in such an environment.

One conventional datacenter technique used for sharing resources is referred to in the art as clustering. Clustering generally involves connecting two or more computers together such that they behave as a single computer to enable high availability of resources. In some cases, load balancing and parallel processing are provided in some clustered environments.

Clustering is generally performed in software (e.g., in the operating system) and allows multiple computers of the cluster to access storage in an organized manner. There are many applications and operating systems that implement clustering techniques such as, for example, the Microsoft Windows NT operating system.

In a conventional cluster configuration, computers (nodes) are coupled by communication links and communicate using a cluster communication protocol. In a traditional clustered machine environment having nodes connected by communication links to form a cluster, each node in the cluster has its own storage adapter (e.g., a Host Bus Adapter (HBA)). These adapters are typically connected to storage entities by one or more communication links or networks. For example, one or more nodes of the cluster may be configured to use storage systems, devices, etc. configured in a storage network (e.g., a Storage Area Network (SAN) connected by a switched fabric (e.g., using FibreChannel)).

In one example, each node HBA includes a unique World Wide Node Name (WWNN) defined within, for example, a FibreChannel (FC) network. The unique WWNN allows storage entities to identify and communicate with each other. To enable cluster coherent storage access, each HBA WWNN needs to be correctly referenced in the storage network. For instance, in a SAN, a Storage Area Network (SAN) zoning configuration is used to control access from HBA resources. Adding nodes to or removing nodes from the cluster, or replacing failed Host Bus Adapters (HBAs) in cluster nodes, requires parallel modifications to the SAN zoning configuration to assure correct storage access.

Summary of the Invention

According to one aspect of the present invention, it is realized that creation of a virtual adapter (e.g., a virtual Host Bus Adapter device (VHBA)) used by one or more nodes in a distributed system (such as, for example, a cluster, grid, multi-node virtual server, etc.) allows just one storage identifier to be assigned. Because one storage identifier is assigned across multiple nodes, the need for modifying a configuration associated with the storage network is eliminated. For example, when software or hardware changes are made in a FC-based storage network, a configuration referred to as a zone configuration may need to be modified so that storage devices may be properly referenced. According to one embodiment of the present invention, a single WWNN may be assigned to a Virtual Host Bus Adapter (VHBA), and underlying hardware and software constructs may be hidden from the operating system and its applications. Because the WWNN is assigned to a virtual adapter which does not change,

storage network zone modification is eliminated when nodes are added or removed from the cluster, grid or multi-processor virtual server.

A virtual adapter according to various embodiments of the present invention may be defined and used, for example, in a conventional cluster-based or grid computing system for accessing storage. In another embodiment of the present invention, such a virtual adapter may be used in a single or multiprocessor computer system. In yet another embodiment of the present invention, such a virtual adapter may be implemented in a Virtual Multiprocessor (VMP) machine. A VMP machine may be, for example, a Symmetric Multiprocessor (SMP) machine, an Asymmetric Multiprocessor (ASMP) machine such as a NUMA machine, or other type of machine that presents an SMP to an operating system or application in a virtualized manner.

In a traditional SMP or cluster environment, high-availability (e.g., using redundant connections) host access to SAN storage requires multiple physical HBAs on each cluster node and high-availability software within the operating system (OS) to manage access of storage resources over multiple paths. Such high-availability software generally requires drivers to be loaded by the OS to enable multi-path I/O (MPIO) based storage access.

By creating a virtual adapter (e.g., a VHBA device) across multiple nodes in a multi-node system (including but not limited to configurations such as a cluster, grid, or VMP) or single node system (e.g., having one or more processors), underlying structures of the underlying multi-path connection to be hidden from the OS. For instance, redundant node interconnects, a FC fabric, and high-availability logic may be isolated from the OS. By isolating the underlying structures from the OS, additional SAN zone configuration is not necessary when changes are made to the underlying hardware (e.g., physical HBAs or other hardware and software). Further, high-availability MPIO drivers are no longer required to be installed and accessed by the operating system.

In a traditional SMP machine, load balancing of storage I/O is also accomplished by adding multiple physical HBAs (i.e., to act as multi-initiators) and software to the operating system to manage the balancing of storage operations across the initiators. According to one embodiment of the invention, in a multi-node machine that involves multiple physical nodes, the operating system running on the one or more of the nodes is provided access to a node local component of the VHBA device. For instance, the node local component of the VHBA device may correspond to a physical HBA device or other physical adapter device that has been abstracted through software. The node local component may be local to a particular node, but

the abstraction, however, allows access to other components (e.g., HBA devices) associated with other nodes in the machine. This abstraction inherently provides multiple-initiator storage access to the operating system on the machine (e.g., multi-node) without additional physical HBAs and operating system software.

According to another embodiment of the present invention, a virtual adapter (e.g., a VHBA device) is used in conjunction with a Virtual Multiprocessor (VMP) machine supported by multiple physical nodes. In such a machine, for example, a single instance of an operating system may be executed across physical nodes. In this case, the single instance of the operating system may be provided access to a node local component of the VHBA device. For example, as discussed above with respect to multi-node systems, the node local component of a VHBA device may correspond to a physical HBA device or other physical adapter device that has been abstracted through software. In the case of a VMP machine, the node local component may be local to a particular node, but the abstraction allows access to other components (e.g., HBA devices) associated with other nodes in the VMP machine. This abstraction inherently provides multiple-initiator storage access to the operating system on the VMP machine (e.g., virtual SMP, virtual ASMP, etc.) without additional physical HBAs and operating system software.

According to one aspect of the invention, a computer is provided which comprises one or more storage entities, at least one of which is capable of servicing one or more requests for access to the one or more storage entities, one or more physical storage adapters used to communicate the one or more requests for access to the one or more storage entities, and a virtual storage adapter adapted to receive the one or more requests and adapted to forward the one or more requests to the one or more physical storage adapters. According to one embodiment, the virtual storage adapter is associated with a virtual server in a virtual computing system. According to another embodiment, the computer system includes a multi-node computer system, at least two nodes of which are adapted to access the virtual storage adapter. According to another embodiment, the virtual storage adapter is identified by a globally unique identifier. According to another embodiment, the unique identifier includes a World Wide Node Name (WWNN) identifier. According to another embodiment, the virtual storage adapter is a virtual host bus adapter (HBA).

According to one embodiment, the computer system further comprises a plurality of communication paths coupling a processor of the computer system and at least one of the one or more storage entities, the virtual storage adapter being capable of directing the one or more

requests over the plurality of communication paths. According to another embodiment, at least one of the one or more requests is translated to multiple request messages being transmitted in parallel over the plurality of communication paths. According to another embodiment, at least one of the plurality of communication paths traverses a switched communication network. According to another embodiment, the switched communication network includes an InfiniBand switched fabric. According to another embodiment, the switched communication network includes a packet-based network.

According to one embodiment, the computer system further comprises a virtualization layer that maps the virtual storage adapter to the one or more physical storage adapters. According to another embodiment, the computer system further comprises a plurality of processors and wherein the virtualization layer is adapted to define one or more virtual servers, at least one of which presents a single computer system interface to an operating system. According to another embodiment, the single computer system interface defines a plurality of instructions, and wherein at least one of the plurality of instructions is directly executed on at least one of the plurality of processors, and at least one other of the plurality of instructions is handled by the virtualization layer. According to another embodiment, the computer system further comprises a plurality of processors, wherein each of the plurality of processors executes a respective instance of a microkernel program, and wherein each of the respective instances of the microkernel program are adapted to communication to cooperatively share access to storage via the virtual storage adapter.

According to one embodiment, the virtual storage adapter is associated with the one or more virtual servers. According to another embodiment, the computer system further comprises a manager adapted to assign the unique identifier to the virtual storage adapter. According to another embodiment, a change in at least one of the one or more physical storage adapters is transparent to the operating system. According to another embodiment, the computer system further comprises configuration information identifying a storage configuration, and wherein a change in at least one of the one or more physical storage adapters is transparent to the operating system. According to another embodiment, the computer system further comprises at least one I/O server, wherein the parallel access requests are serviced in parallel by the I/O server.

According to one embodiment, the at least one of the one or more storage entities receives the multiple request messages and services the multiple request messages in parallel. According to another embodiment, the virtual storage adapter is associated with a node in a

multi-node computing system. According to another embodiment, the multi-node computing system is a grid-based computing system. According to another embodiment, the multi-node computing system is a cluster-based computing system. According to another embodiment, the virtual storage adapter is associated with a single computer system. According to another embodiment, the multi-node computing system supports a virtual computing system that executes on the multi-node computing system, and wherein the virtual computing system is adapted to access the virtual storage adapter. According to another embodiment, the single computer system supports a virtual computing system that executes on the single computer system, and wherein the virtual computing system is adapted to access the virtual storage adapter.

According to one embodiment, the virtual storage adapter is identified by a globally unique identifier. According to another embodiment, the globally unique identifier includes a World Wide Node Name (WWNN) identifier. According to another embodiment, the virtual storage adapter is identified by a globally unique identifier. According to another embodiment, the globally unique identifier includes a World Wide Node Name (WWNN) identifier.

According to another aspect of the present invention, a computer-implemented method is provided in a computer system having one or more storage entities, at least one of which is capable of servicing one or more requests for access to the one or more storage entities, and having one or more physical storage adapters used to communicate the one or more requests for access to the one or more storage entities. The method comprises an act of providing for a virtual storage adapter, the virtual adapter adapted to perform acts of receiving the one or more requests, and forwarding the one or more requests to the one or more physical storage adapters.

According to one embodiment, the method further comprises an act of associating the virtual storage adapter with a virtual server in a virtual computing system. According to another embodiment, the computer system includes a multi-node computer system, and wherein at least two nodes of the computer system each perform an act of accessing the virtual storage adapter. According to another embodiment, the method further comprises an act of identifying the virtual storage adapter by a globally unique identifier. According to another embodiment, the act of identifying the virtual storage adapter includes an act of identifying the virtual storage adapter by a World Wide Node Name (WWNN) identifier. According to another embodiment, the act of providing for a virtual storage adapter includes an act of providing a virtual host bus adapter (HBA). According to another embodiment, the computer system further comprises a plurality of communication paths coupling a processor of the

computer system and at least one of the one or more storage entities, and wherein the method further comprises an act of directing, by the virtual storage adapter, the request over the plurality of communication paths.

According to one embodiment, the computer system further comprises a plurality of communication paths coupling a processor of the computer system and at least one of the one or more storage entities, and wherein the method further comprising acts of translating at least one of the one or more requests to multiple request messages and transmitting the multiple request messages in parallel over the plurality of communication paths. According to another embodiment, at least one of the plurality of communication paths traverses a switched communication network. According to another embodiment, the switched communication network includes an InfiniBand switched fabric. According to another embodiment, the switched communication network includes a packet-based network. According to another embodiment, the method further comprises an act of mapping the virtual storage adapter to the one or more physical storage adapters.

According to one embodiment, the act of mapping is performed in a virtualization layer of the computer system. According to another embodiment, the computer system further comprises a plurality of processors, and wherein the method further comprises an act of defining one or more virtual servers, at least one of which presents a single computer system interface to an operating system. According to another embodiment, the computer system further comprises a plurality of processors, and wherein the method further comprises an act of defining one or more virtual servers, at least one of which presents a single computer system interface to an operating system. According to another embodiment, the act of defining is performed by the virtualization layer. According to another embodiment, the act of defining is performed by the virtualization layer. According to another embodiment, the single computer system interface defines a plurality of instructions, and wherein the method further comprises an act of executing at least one of the plurality of instructions directly on at least one of the plurality of processors, and handling, by the virtualization layer, at least one other of the plurality of instructions.

According to one embodiment, the computer system comprises a plurality of processors, and wherein each of the plurality of processors performs an act of executing a respective instance of a microkernel program, and wherein each of the respective instances of the microkernel program communicate to cooperatively share access to storage via the virtual storage adapter. According to another embodiment, the method further comprises an act of

associating the virtual storage adapter with the one or more virtual servers. According to another embodiment, the computer system further comprises a manager, and wherein the method further comprises an act of assigning, by the manager, the unique identifier to the virtual storage adapter. According to another embodiment, a change in at least one of the one or more physical storage adapters is transparent to the operating system. According to another embodiment, the method further comprises an act of maintaining configuration information identifying a storage configuration, and wherein a change in at least one of the one or more physical storage adapters is transparent to the storage configuration.

According to one embodiment, the computer system further comprises at least one I/O server, wherein the parallel access request messages are serviced in parallel by the I/O server. According to another embodiment, the method further comprises acts of receiving, by the at least one of the one or more storage entities, the multiple request messages, and servicing the multiple request messages in parallel. According to another embodiment, the method further comprises an act of associating the virtual storage adapter with a node in a multi-node computing system. According to another embodiment, the multi-node computing system is a grid-based computing system. According to another embodiment, the multi-node computing system is a cluster-based computing system. According to another embodiment, the method further comprises an act of associating the virtual storage adapter with a single computer system.

According to one embodiment, the multi-node computing system supports a virtual computing system that executes on the multi-node computing system, and wherein the method further comprises an act of accessing, by the virtual computing system, the virtual storage adapter. According to another embodiment, the single computer system supports a virtual computing system that executes on the single computer system, and wherein the method further comprises an act of accessing, by the virtual computing system, the virtual storage adapter. According to another embodiment, the method further comprises an act of identifying the virtual storage adapter by a globally unique identifier. According to another embodiment, the act of identifying the virtual storage adapter includes an act of identifying the virtual storage adapter by a World Wide Node Name (WWNN) identifier. According to another embodiment, the method further comprises an act of identifying the virtual storage adapter by a globally unique identifier. According to another embodiment, the globally unique identifier includes a World Wide Node Name (WWNN) identifier.

Further features and advantages of the present invention as well as the structure and operation of various embodiments of the present invention are described in detail below with reference to the accompanying drawings. In the drawings, like reference numerals indicate like or functionally similar elements. Additionally, the left-most one or two digits of a reference numeral identifies the drawing in which the reference numeral first appears.

BRIEF DESCRIPTION OF DRAWINGS

The accompanying drawings are not intended to be drawn to scale. In the drawings, each identical or nearly identical component that is illustrated in various figures is represented by a like numeral. For purposes of clarity, not every component may be labeled in every drawing. In the drawings:

Figure 1 is a block diagram of a virtual server architecture according to one embodiment of the present invention;

Figure 2 is a block diagram of a system for providing virtual services according to one embodiment of the present invention;

Figure 3 is a block diagram showing a mapping relation between virtual processors and physical nodes according to one embodiment of the present invention;

Figure 4 is a block diagram showing scheduling of virtual processor tasks according to one embodiment of the present invention;

Figure 5 is a block diagram showing scheduling of virtual processor tasks in accordance with another embodiment of the present invention;

Figure 6 is a block diagram showing an example memory mapping in a virtual server system in accordance with another embodiment of the present invention;

Figure 7 is a block diagram showing an example execution level scheme in accordance with another embodiment of the present invention;

Figure 8 is a block diagram showing an example distributed virtual machine monitor architecture in accordance with another embodiment of the present invention;

Figure 9 is a block diagram showing an example system architecture upon which a virtual computing system in accordance with another embodiment of the present invention may be implemented; and

Figure 10 is a block diagram showing a virtual storage architecture according to one embodiment of the present invention.

Detailed Description

In one aspect, a virtualized storage adapter architecture is provided wherein lower level details of the storage adapter architecture are isolated from the operating system and application executing on the computing system. That is, the storage adapter used to access storage is virtualized. Such a virtual storage adapter architecture contrasts to conventional virtual storage architectures where actual storage entities (e.g., volumes, disks, etc., but not the adapters used to access such entities) are virtualized. Isolation from the operating system and applications may be performed, for example, by providing a virtual storage adapter that is backed by one or more physical adapters.

Such a virtualized storage adapter architecture may be used with a single or multinode computing system as discussed above. For instance, a virtual storage architecture may be implemented in cluster-based or grid computing systems. Further, various aspects of the present invention may be implemented in a virtual computing system as discussed in further detail below. However, it should be appreciated that a virtual storage adapter architecture may be used with any computing architecture (e.g, single node, multi-node, cluster, virtual, VMP, etc.), and the invention is not limited to any computer system type or architecture. An example virtual storage architecture according to one embodiment of the present invention is discussed below with more particularity in reference to Figure 10.

According to another embodiment of the present invention, a horizontal virtualization architecture is provided wherein applications are distributed across virtual servers, and the horizontal virtualization architecture is capable of accessing storage through a virtual storage adapter. In one example system, an application is scaled horizontally across at least one virtual server, comprised of a set of virtual processors, each of which is mapped to one or more physical nodes. From the perspective of the application, the virtual server operates like a shared memory multi-processor, wherein the same portion of the application is located on one or more of the virtual processors, and the multiple portions operate in parallel. The resulting system allows applications and operating systems to execute on virtual servers, where each of these virtual servers span a collection of physical servers (or nodes) transparent to the applications and operating systems. That is, the virtual server presents, to the operating system and application a single system where single instance of an operating system runs. Such a system according to one embodiment is contrasted by conventional clustered computing systems that support single system image as typically understood in the art, in that multiple instances of an operating system are clustered to create an illusion of a single system to the

application programmers. Further, such a system according to one embodiment is unlike conventional "grid" computing systems as typically understood in the art, as no application modifications are required for the applications to execute on the virtualization architecture.

Figure 1 shows one example system 101 that may be used to execute one or more data center applications. System 101 may include one or more system layers providing layers of abstraction between programming entities. As discussed above, a virtualization layer 104 is provided that isolates applications on a guest operating system (GOS) operating in layers 102 and 103, respectively, from an underlying hardware layer 105. Such applications may be, for example, any application program that may operate in a data center environment. For instance, a database server application, web-based application, e-mail server, file server, or other application that provides resources to other systems (e.g., systems 107A-107C) may be executed on system 101. Such applications may communicate directly with virtualization layer 104 (e.g., in the case of a database server application, wherein the application is part of the operating system) or may communicate indirectly through operating system layer 103. Virtualization layer 104 in turn maps functions performed by one or more virtual processors to functions performed by one or more physical entities in hardware layer 105. These entities may be, for instance, physical nodes having one or more processors.

In one aspect, virtualization layer 104 presents, to application layer 102 and operating system layer 103 a single system presented in the form of a virtual server. In one embodiment, a single instance of an OS is executed by the virtual server. In particular, a distributed virtual machine monitor creates a single system image, upon which a single instance of a virtual server is executed. The virtual server acts as a single system, executing a single instance of the OS. This architecture contrasts to conventional clustering systems where multiple OS entities executing on multiple systems cooperate to present a single system (e.g., to an application programmer that develops programs to be executed on a clustered OS). According to another embodiment of the present invention, this virtual server includes one or more constructs similar to a physical server (storage, memory, I/O, networking), but these constructs are virtual and are mapped by virtualization layer 104 to one or more hardware entities.

Physical entities may communicate with each other over an interconnect (not shown) for the purpose of sharing access to resources within hardware layer 105. For instance, a distributed memory architecture may be used to allow hardware devices (e.g., nodes to share other non-local memory. Other hardware entities (e.g., network, storage, I/O, etc.) may also be shared by nodes through an interconnect.

System 101 may be coupled to one or more external communication networks (e.g., network 106) for the purpose of sharing resources with one or more systems (e.g., systems 107A-107C). System 101 may function as part of an overall computing system 100 to perform one or more tasks. For instance, system 100 may function as a client-server, n-tiers, or other type of architecture that executes one or more applications in a cooperative system. It should be appreciated that system 100 may include any number and type of computing systems, architecture, application, operating system or network, and the invention is not limited to any particular one(s).

Example Architecture

Figure 2 shows an example architecture of a system 201 according to one embodiment of the invention. System 201 includes an upper layer 202 including one or more operating systems 207A-207C executed by one or more virtual servers 208A-208C, respectively. According to one embodiment, virtual servers 208A-208C present, to their respective operating systems 207A-207C, single system regardless of the number of hardware nodes (e.g., nodes 210A-210D) included in a particular virtual server.

Operating systems 207A-207C may be, for example, commodity operating systems that may be ported to a Virtual Machine Architecture (VMA) presented by a distributed virtual machine monitor. A virtual server may be an instance of an architecture presented by a virtualization layer (e.g., layer 104). A virtual server may have a persistent identity and defined set of resource requirements (e.g., storage, memory, and network) resource access privileges, and/or resource limits.

Distributed virtual machine monitor (or DVMM) 203 provides an abstraction layer for mapping resources presented by each virtual server to other upper layer 202 programs to underlying hardware 204. In one embodiment, DVMM 203 includes one or more microkernel 209A-209E, each of which are pseudo-machines, each of which runs on a single node and manages the resources associated with that node. Each microkernel 209A-209E may include a virtual memory which it manages, this memory space spanning one or more portions of available physical memory associated with participating nodes.

Hardware layer 204 may include, for example, one or more nodes 210A-210E coupled by a network 211. These nodes may be, for example, general-purpose processing systems having one or more physical processors upon which tasks are performed.

According to one embodiment, an organizational concept of a frame may be defined, the frame identifying a set of nodes and other hardware entities that may be used to operate as an organizational unit. Elements within the frame may be capable of communicating between each other over a network 211. In one example, network 211 may include a low-latency high-bandwidth communication facility (e.g., InfiniBand, PCI-Express, GigiNet, Ethernet, Gigabit Ethernet, 10 Gigabit Ethernet, etc.). However, it should be appreciated that the invention is not limited to low-latency communication facility, as other communication methods may be used. Network 211 may also include one or more elements (e.g., switching or routing elements) that create an interconnected frame.

In one embodiment, nodes (e.g., nodes 210A-210E) are restricted to participating in one and only one frame. A defined frame and its associated hardware may be associated with a distributed server, and the entities of that frame may perform the physical operations associated with that virtual distributed server.

In one embodiment, a distributed server is a collection of software and hardware components. For example, hardware components may include commodity servers coupled to form a cluster. Software associated with each distributed server runs on this cluster and presents a multi-processor system architecture two upper layers, defining a virtual server that is capable of hosting a guest operating system (GOS). Components of a distributed server may include a distributed virtual machine monitor program, interconnects, processors, memory, I/O devices and software and protocols used to bind them. A guest operating system (GOS), such as, for example, UNIX (e.g., Linux, SUSE, etc.), Microsoft Windows Server, or other operating system executes upon the virtual server. In one embodiment, the guest operating system operates as if it was running on a non-cluster multi-processor system having coherent shared memory.

System 201 may also include a manager 212 that manages the configuration of system 201. Manager 212 may include an associated management database 213 that stores information relating to the configuration of system 201. Manager 212 may also communicate with a management agent (not shown) executed by one or more virtual servers of system 201 for the purpose of performing configuration changes, monitoring performance, and performing other administrative functions associated with system 201. The following section discusses an example management architecture for managing a virtual computing architecture, and various advantages of a scalable virtual computing system according to various embodiments of the

present invention.

Management Architecture

As discussed above, the virtualization architecture allows for an expansion (or a contraction) of resources used by an executing virtual computing system. Such expansion or contraction may be needed from time to time as customer and business needs change. Also, applications or the operating systems themselves may need additional (or less) resources as their requirements change (e.g., performance, loading, etc.). To this end, a capability may be provided for changing the amount and allocation of resources, both actual and virtual, to the virtual computing system. More specifically, additional resources (e.g., nodes, network, storage, I/O, etc.) may be allocated (or deallocated) in real time to a frame and these resources may then be used (or not used) by a distributed server. Similarly, virtualized resources (e.g., virtual processors, virtual I/O, virtual networking, etc.) as well as physical resources may be allocated or deallocated to a virtual server. In this manner, the virtual computing system may be scaled up/scaled down as necessary.

The ability for allocating or deallocating resources may be provided using, for example, manager 212 and one or more management agents. Such a system is described with more particularity in the co-pending U.S. patent application filed April 26, 2004 entitled "METHOD AND APPARATUS FOR MANAGING VIRTUAL SERVERS" under Attorney Docket Number K2000-700100, which is incorporated by reference in its entirety.

According to one aspect of the present invention, a management capability is provided for a virtual computing platform. This platform allows scale up and scale down of virtual computing systems, and such a management capability provides for control of such scale up and scale down functions. For instance, a capability is provided to allocate and/or deallocate resources (e.g., processing, memory, networking, storage, etc.) to a virtual computing system. Such control may be provide, for example, to an administrator through an interface (e.g., via a CLI, or GUI) or to other programs (e.g., via a programmatic interface).

According to one aspect of the present invention, an interface is provided that allows for the addition or removal of resources during the execution of a virtual computing system. Because resource allocation may be changed without restarting the virtual computing system, a flexible tool is provided for administrators and programs for administering computing resources.

In the case where such a virtual computing system is provided in a datacenter, an

administrator may be capable of provisioning resources in real time to support executing virtual servers. Conventionally, data center server resources are hard-provisioned, and typically require interruption of server operation for resources to be changed (e.g., change in memory, network, or storage devices).

According to one embodiment of the present invention, a virtual computing system is provided that allows a network administrator to provision computing resources in real-time ("on-the-fly") without a restart of a virtual computing system. For instance, the administrator may be presented an interface through which resources may be allocated to a virtual server (e.g., one that emulates a virtual multiprocessor computer). The interface may display a representation of an allocation of physical resources and mapping to virtual resources used by a virtual server. For example, the interface may provide an ability to map virtual servers to sets of physical resources, such as a virtual processor that is mapped to a physical processor.

According to another embodiment, a capability is provided to allocate and/or deallocate resources (e.g., processing, memory, networking, storage, etc.) to a virtual computing system. Such control may be provide, for example, to an administrator through an interface (e.g., via a CLI, or GUI) or to other programs (e.g., via a programmatic interface). According to another embodiment, an interface is provided that allows for the addition or removal of resources during the execution of a virtual computing system. Because resource allocation may be changed without restarting the virtual computing system, a flexible tool is provided for administrators and programs for administering computing resources. This tool permits an administrator to grow or shrink the capabilities of a virtual server system graphically or programmatically.

For instance, the administrator may be presented an interface through which resources may be allocated to a virtual server (e.g., one that emulates a virtual multiprocessor computer). The interface may display a representation of an allocation of physical resources and mapping to virtual resources used by a virtual server. For example, the interface may provide an ability to map virtual servers to sets of physical resources, such as a virtual processor that is mapped to a physical processor. In one embodiment, a virtual server can span a collections of a physical nodes coupled by an interconnect. This capability allows, for example, an arbitrarily-sized virtual multiprocessor system (e.g., SMP, Numa, ASMP, etc.) to be created.

Such capabilities may be facilitated by a management agent and server program that collectively cooperates to control configuration of the virtual and distributed servers. According to one embodiment, the management server writes information to a data store to

indicate how each node should be configured into virtual and distributed servers. Each management agent may then read the data store to determine its node's configuration. The configuration may be, for example, pushed to a particular management agent, pulled from the management server by the management agent, or a combination of both techniques. The management agent may pass this information to its virtual machine monitor program which uses the information to determine the other nodes in its distributed server with whom it is tasked to cooperatively execute a set of virtual servers.

An administrator or other program may, using one or more interfaces (e.g., UI, CLI, programmatic, etc.) to allocate or deallocate resources to virtual servers or distributed servers. More particularly, the interface may allow an administrator or program to associate a hardware resource (e.g., an I/O device, network interface, node having one or more physical processors, etc.) to a distributed server of a frame. As discussed further below with reference to Figure 3, a frame (e.g., frame 302A, 302B) may define a partitioned set of hardware resources, each of which sets may form multiple distributed servers, each of which sets may be associated with one or more virtual servers. Alternatively, a hardware resource may be allocated directly to a virtual server.

A hardware device may be unassigned to a particular distributed server within a frame in which the hardware device is coupled, for example, during initial creation of the distributed server (e.g., with unassigned resources), by adding new hardware to the frame, or by virtue of having previously unassigning the hardware resource to a distributed server or virtual server. Such unassigned resources may be, for example, grouped into a "pool" of unassigned resources and presented to an administrator or program as being available for assignment. Once assigned, the virtual computing system may maintain a representation of the assignment (or association) in a data structure (e.g., in the data store described above) that relates the hardware resource to a particular distributed server or virtual server.

Once an actual resource (e.g., hardware) is assigned, virtual resources associated with the hardware resource may be defined and allocated to virtual servers. For instance, one or more VNICs (virtual network interface cards) may be defined that can be backed by one or more actual network interface devices. Also, a new node may be assigned to a partition upon which a virtual server is executed, and any CPUs of the newly-assigned nodes may be assigned as additional virtual processors (VPs) to the virtual server.

In one example, the management server may use an object model to manage components (e.g., resources, both physical and virtual) of the system. Manageable objects and

object collections may be defined along with their associations to other manageable objects. These objects may be stored in a data structure and shared with other management servers, agents, or other software entities. The management architecture may implement a locking mechanism that allows orderly access to configurations and configuration changes among multiple entities (administrators, programs, etc.).

According to one embodiment, a management agent at each node interacts with the virtual machine monitor program and with outside entities, such as, for example, a management server and a data store. In one example, the management server provides command and control information for one or more virtual server systems. The management agent acts as the virtual machine monitor program tool to communicate with the management server, and implement the actions requested by the management server. In one example, the management agent is a virtual machine monitor user process. According to another embodiment, the data store maintains and provides configuration information upon demand. The data store may reside on the same or different node as the management server, or may be distributed among multiple nodes.

The management agent may exist within a constrained execution environment, such that the management agent is isolated from both other virtual server processes as well as the virtual machine monitor program. That is, the management agent may not be in the same processor protection level as the rest of the virtual machine monitor program. Alternatively, the management agent may operate at the same level as the virtual machine monitor program or may form an integral part of the virtual machine monitor program. In one embodiment, the management agent may be responsible for a number of tasks, including configuration management of the system, virtual server management, logging, parameter management, and event and alarm propagation.

According to one embodiment, the virtual machine monitor management agent may be executed as a user process (e.g., an application on the virtual server), and therefore may be scheduled to be executed on one or more physical processors is similar to an application. Alternatively, the management agent may be executed as an overhead process at a different priority than an application. However, it should be appreciated that the management agent may be executed at any level of a virtual computing system hierarchy and at any protection or priority level.

According to one embodiment, interactions between the management agent and the management server may be categorized as either command or status interactions. According to

one embodiment, commands originate with the management server and are sent to the management agent. Commands include, but are not limited to, distributed server operations, instructions to add or remove a node, processor, memory and/or I/O device, instructions to define or delete one or more virtual servers, a node configuration request, virtual server operations, status and logging instructions, heartbeat messages, alert messages, and other miscellaneous operations. These commands or status interactions may be transmitted, for example, using one or more communication protocols (e.g., TCP, UDP, IP or others). It should be appreciated that the virtual computing platform may be managed using a different architecture, protocols, or methods, and it should be understood that the invention is not limited to any particular management architecture, protocols, or methods.

Mapping of Virtual Servers

Figure 3 shows in more detail an example mapping of one or more virtual servers to a grouping of hardware referred to hereinafter as a partition according to one embodiment of the invention. A collection of one or more virtual processors is arranged in a set. In one embodiment, a virtual server (VS) may be viewed as a simple representation of a complete computer system. A VS, for example, may be implemented as a series of application programming interfaces (APIs). An operating system is executed on a virtual server, and a distributed virtual machine monitor may manage the mapping of VPs onto a set of physical processors. A virtual server (e.g., VS 301A-301E) may include one or more VPs (e.g., 303A-303C), and the number of VPs in a particular VS may be any number.

Hardware nodes and their associated resources are grouped together into a set referred to herein as a frame. According to one embodiment, a virtual server is associated with a single frame, and more than one virtual server may be serviced by a frame. In the physical realm, nodes (e.g., nodes 304A-304C) may be associated with a particular frame (e.g., frame 302A). In one example, a frame (e.g., frame 302A, 302B) may define a partitioned set of hardware resources, each of which sets may form multiple distributed servers, each of which sets may be associated with one or more virtual servers. In one embodiment, virtual processors are mapped to physical processors by the distributed virtual machine monitor. In one embodiment, there may be a one-to-one correspondence between virtual processors and physical processors. Nodes within a frame may include one or more physical processors upon which virtual processor tasks may be scheduled. Although several example mappings are shown, it should

be appreciated that the invention is not limited to the shown mappings. Rather, any mapping may be provided that associates a virtual server to a frame.

However, there may be configurations that are not allowed for reasons having to do with security, performance, or other reasons. For instance, according to one embodiment, mapping of a virtual server to more than one frame may not be permitted (e.g., nodes outside of a frame are not connected to the internal frame interconnect). Other configurations may not be permitted based on one or more rules. For instance, in one example, a physical processor may not be permitted to be allocated to more than one distributed server. Also, the number of active physical processors in use may not be permitted to be less than the number of virtual processors in the virtual processing system. Other restriction rules may be defined alone or in combination with other restriction rules.

Scheduling

Figure 4 shows an example scheduling relation between virtual processors and physical processors according to one embodiment of the invention. As shown, virtual server 401 includes two virtual processors VP 403A-403B. Each of these VPs are mapped to nodes 404A-404B, respectively in frame 402. Node 404A may include one processor 405A upon which a task associated with VP 403A may be scheduled.

There may be a scheduler within the distributed virtual machine monitor that handles virtual processor scheduling. In one example, each virtual processor is mapped to one process or task. The scheduler may maintain a hard affinity of each scheduled process (a VP) to a real physical processor within a node. According to one embodiment, the distributed virtual machine monitor may execute one task per virtual processor corresponding to its main thread of control. Tasks in the same virtual server may be simultaneously scheduled for execution.

Figure 5 shows a more detailed example showing how virtual server processes may be scheduled according to one embodiment of the present invention. In the example, there are four virtual servers, VS1 (item 501), VS2 (item 502), VS3 (item 503), and VS4 (item 504) defined in the system. These virtual servers have one or more virtual processors (VPs) associated with them.

These four virtual processors are mapped to two nodes, each of which nodes includes two physical processors, P1-P4. The distributed virtual machine monitor maps each virtual server to an individual process. Each virtual processor (VP) within a virtual server is a thread within this process. These threads may be, for example, bound via hard affinity to a specific

physical processor. To the distributed virtual machine monitor, each of the virtual servers appears as a process running at a non-privileged level. Each of the individual virtual processors included in a virtual server process are component threads of this process and may be scheduled to run on a separate, specific physical processor.

With the example configuration having two dual processor nodes (four physical processors total), in one embodiment of the invention there may be up to a maximum of four VPs created in any virtual server. Also, with a total number of eight VPs, there are eight threads. As shown in Figure 5, the distributed virtual machine monitor may run each virtual server process at approximately the same time (e.g., for performance reasons as related processes running at different times may cause delays and/or issues relating to synchronization). That is, the VS4 processes are scheduled in one time slot, VS3 processes in the next, and so forth. There may be "empty" processing slots in which management functions may be performed or other overhead processes. Alternatively, the scheduler may rearrange tasks executed in processor slots to minimize the number of empty processor slots.

Further, the scheduler may allow for processors of different types and/or different processing speeds to perform virtual server tasks associated with a single virtual server. This capability allows, for example, servers having different processing capabilities to be included in a frame, and therefore is more flexible in that an administrator can use disparate systems to construct a virtual computing platform. Connections between different processor types are facilitated, according to one embodiment, by not requiring synchronous clocks between processors.

Memory

Figure 6 shows a block diagram of a memory mapping in a virtual computer system according to one embodiment of the invention. In general, the distributed virtual machine monitor may make memory associated with hardware nodes available to the guest operating system (GOS) and its applications. The distributed virtual machine monitor (DVMM), through a virtual machine architecture interface (hereinafter referred to as the VMA), offers access to a logical memory defined by the distributed virtual machine monitor and makes available this memory to the operating system and its applications.

According to one embodiment, memory is administered and accessed through a distributed memory manager (DMM) subsystem within the distributed virtual machine monitor. Memory may, therefore, reside on more than one node and may be made available to

all members of a particular virtual server. However, this does not necessarily mean that all memory is distributed, but rather, the distributed virtual machine monitor may ensure that local memory of a physical node is used to perform processing associated on that node. In this way, local memory to the node is used when available, thereby increasing processing performance. One or more "hint" bits may be used to specify when local memory should be used, so that upper layers (e.g., virtual layers) can signal to lower layers when memory performance is critical.

Referring to Figure 6 and describing from left to right, a node's physical memory 601 may be arranged as shown in Figure 6, where a portion of the node's physical memory is allocated to virtual memory 602 of the distributed virtual machine monitor memory. As shown, distributed memory associated with the node may be part of a larger distributed memory 603 available to each distributed server. Collectively, the distributed memories of each node associated with the distributed server may be made available to a virtual server as logical memory 604 and to the operating system (GOS), as if it were a physical memory. Memory 604 is then made available (as process virtual memory 605) to applications.

GOS page table manipulation may, for example, be performed by the distributed virtual machine monitor in response to GOS requests. Because, according to one embodiment, the GOS is not permitted direct access to page tables to ensure isolation between different virtual servers, the distributed virtual machine monitor may be configured to perform page table manipulation. The distributed virtual machine monitor may handle all page faults and may be responsible for virtual address spaces on each virtual server. In particular, the DMM subsystem of the distributed virtual machine monitor (DVMM) may perform operations on page tables directly.

Memory operations that may be presented to the operating system through the virtual machine architecture (VMA). According to one embodiment of the present invention, the VMA may include memory operations that are similar in function to that of conventional architecture types (e.g., Intel). In this manner, the amount of effort needed to port a GOS to the VMA is minimized. However, it should be appreciated that other architecture types may be used.

In the case where the architecture is an Intel-based architecture, memory operations that may be presented include management of physical and logical pages, management of virtual address spaces, modification of page table entries, control and modification of base registers, management of segment descriptors, and management of base structures (e.g., GDT (global

descriptor table), LDT (local descriptor table), TSS (task save state) and IDT (interrupt dispatch table)).

According to one embodiment, access to such memory information may be isolated. For instance, access to hardware tables such as the GDT, LDT, and TSS may be managed by the VMA. More particularly, the VMA may maintain copies of these tables for a particular virtual server (providing isolation), and may broker requests and data changes, ensuring that such requests and changes are valid (providing additional isolation). The VMA may provide as a service to the GOS access to instructions and registers that should not be accessed at a privileged level. This service may be performed by the VMA, for example, by a function call or by transferring data in a mapped information page.

It can be appreciated that although the VMA may expose logical memory to the GOS, actual operations may be performed on memory located in one or more physical nodes. Mapping from virtual to logical memory may be performed by the VMA. For instance, a virtual address space (or VAS) may be defined that represents a virtual memory to logical memory mapping for a range of virtual addresses.

Logical memory may be managed by the GOS, and may be allocated and released as needed. More particularly, the GOS may request (e.g., from the VMA) for an address space to be created (or destroyed) through the VMA, and the DMM subsystem of the DVMM may perform the necessary underlying memory function. Similarly, the VMA may include functions for mapping virtual addresses to logical addresses, performing swapping, perform mapping queries, etc.

Remote Direct Memory Access (RDMA) techniques may also be used among the nodes to speed memory access among the nodes. Remote Direct Memory Access (RDMA) is a well-known network interface card (NIC) feature that lets one computer directly place information into the memory of another computer. The technology reduces latency by minimizing demands on bandwidth and processing overhead.

Input/Output

Regarding I/O, the VMA may provide isolation between the GOS and distributed virtual machine monitor. According to one embodiment of the present invention, the VMA functions as a thin conduit positioned between the GOS and a DVMM I/O subsystem, thereby providing isolation. In one embodiment, the GOS is not aware of the underlying hardware I/O devices and systems used to support the GOS. Because of this, physical I/O devices may be

shared among more than one virtual server. For instance, in the case of storage I/O, physical storage adapters (e.g., HBAs, IB HCA with access to TCA I/O Gateway) may be shared among multiple virtual servers.

In one implementation, GOS drivers associated with I/O may be modified to interface with the VMA. Because the size of the distributed virtual machine monitor should, according to one embodiment, be minimized, drivers and changes may be made in the GOS, as there is generally more flexibility in changing drivers and configuration in the GOS than the distributed virtual machine monitor.

I/O functions that may be performed by the distributed virtual machine monitor in support of the GOS may include I/O device configuration and discovery, initiation (for both data movement and control), and completion. Of these types, there may be varying I/O requests and operations specific to each type of device, and therefore, there may be one or more I/O function codes that specify the functions to be performed, along with a particular indication identifying the type of device upon which the function is performed. I/O support in the VMA may act as a pipe that channels requests and results between the GOS and underlying distributed virtual machine monitor subsystem.

I/O devices that may be shared include, for example, FibreChannel, InfiniBand and Ethernet. In hardware, I/O requests may be sent to intelligent controllers (referred to hereinafter as I/O controllers) over multiple paths (referred to as multipathing). I/O controllers service the requests by routing the request to virtual or actual hardware that performs the I/O request possibly simultaneously on multiple nodes (referred to as multi-initiation), and returns status or other information to the distributed virtual machine monitor.

In one example I/O subsystem, the distributed virtual machine monitor maintains a device map that is used to inform the GOS of devices present and a typing scheme to allow access to the devices. This I/O map may be an emulation of a bus type similar to that of a conventional bus type, such as a PCI bus. The GOS is adapted to identify the device types and load the appropriate drivers for these device types. Drivers pass specific requests through the VMA interface, which directs these requests (and their responses) to the appropriate distributed virtual machine monitor drivers.

The VMA configuration map may include, for example, information that allows association of a device to perform an operation. This information may be, for example, an index/type/key information group that identifies the index of the device, the device type, and

the key or instance of the device. This information may allow the GOS to identify the I/O devices and load the proper drivers.

Once the GOS has determined the I/O configuration and loaded the proper drivers, the GOS is capable of performing I/O to the device. I/O initiation may involve the use of the VMA to deliver an I/O request to the appropriate drivers and software within the distributed virtual machine monitor. This may be performed, for example, by performing a call on the VMA to perform an I/O operation, for a specific device type, with the request having device-specific codes and information. The distributed virtual machine monitor may track which I/O requests have originated with a particular virtual server and GOS. I/O commands may be, for example, command/response based or may be performed by direct CSR (command status register) manipulation. Queues may be used between the GOS and distributed virtual machine monitor to decouple hardware from virtual servers and allow virtual servers to share hardware I/O resources.

According to one embodiment of the present invention, GOS drivers are virtual port drivers, presenting abstracted services including, for example, send packet/get packets functions, and write buffer/read buffer functions. In one example, the GOS does not have direct access to I/O registers. Higher level GOS drivers, such as class drivers, filter drivers and file systems utilize these virtual ports.

In one embodiment of the present invention, three different virtual port drivers are provided to support GOS I/O functions: console, network and storage. These drivers may be, for example, coded into a VMA packet/buffer interface, and may be new drivers associated with the GOS. Although a new driver may be created for the GOS, above the new driver the GOS kernel does not access these so called "pass-through" virtual port drivers and regular physical device drivers as in conventional systems. Therefore, virtual port drivers may be utilized within a context of a virtual system to provide additional abstraction between the GOS and underlying hardware.

According to another embodiment, the use of virtual port drivers may be restricted to low-level drivers in the GOS, allowing mid-level drivers to be used as is (e.g., SCSI multi-path drivers). With respect to the I/O bus map, virtual port drivers are provided that present abstracted hardware vs. real hardware (e.g., VHBA v. HBA devices), allowing the system (e.g., the distributed virtual machine monitor) to change the physical system without changing the bus map. Therefore, the I/O bus map has abstraction as the map represents devices in an abstract sense, but does not represent the physical location of the devices. For example, in a

conventional PC having a PCI bus and PCI bus map, if a board in the PC is moved, the PCI map will be different. In one embodiment of the present invention, a system is provided wherein if the location of a physical device changes, the I/O map presented to higher layers (e.g., application, GOS) does not change. This allows, for example, hardware devices/resources to be removed, replaced, upgraded, etc., as the GOS does not experience a change in "virtual" hardware with an associated change in actual hardware.

Example I/O Function

The following is an example of an I/O function performed in a virtual server as requested by a GOS (e.g., Linux). The I/O function in the example is initially requested of the Guest Operating System. For instance, a POSIX-compliant library call may invoke a system service that requests an I/O operation.

The I/O operation passes through a number of layers including, but not limited to:

- Common GOS I/O processing. A number of common steps might occur including request aggregation, performance enhancements and other I/O preprocessing functions. The request may be then passed to a first driver level referred to as an "Upper Level" driver.
- "Upper Level" drivers that are not in direct hardware contact, but provide support for a particular class of devices. The request is further processed here and passed on to Lower Level drivers.
- " Lower Level" drivers are in direct hardware contact. These drivers are specific to a virtual server and are modified to work in direct contact with the VMA I/O interface as discussed above. These drivers process the request and pass the request to the VMA I/O component as if the I/O component was a specific hardware interface.
- The VMA I/O component routes the request to the proper distributed virtual machine monitor (DVMM) drivers for processing.
- The DVMM I/O layer now has the request and processes the request as needed. In this example, a set of cooperating drivers moves the request onto network drivers (e.g., InfiniBand drivers) and out onto the hardware (e.g., storage adapters, network interfaces, etc.).

In a virtual server according to one embodiment, all processors may initiate and complete I/O operations concurrently. All processors are also capable of using multipath I/O to direct I/O requests to the proper destinations, and in turn each physical node can initiate its own I/O requests. Further, the network (e.g., an interconnect implementing InfiniBand) may offer storage devices (e.g., via FibreChannel) and networking services (e.g., via IP) over the

network connection (e.g., an InfiniBand connection). This set of capabilities provides the distributed virtual machine monitor, and therefore, virtual servers, with a very high performance I/O system. An example architecture that shows some of these concepts is discussed further below with reference to Figure 9. A specific virtual architecture that shows these concepts as they relate to storage is discussed further below with reference to Figure 10.

Interrupts and Exceptions

Other interfaces to the GOS may also provide additional isolation. According to one aspect of the present invention, interrupts and exceptions may be isolated between the GOS and distributed virtual machine monitor (DVMM). More particularly, interrupts and exceptions may be handled, for example, by an interface component of the VMA that isolates the GOS from underlying interrupt and exception support performed in the DVMM. This interface component may be responsible for correlation and propagation of interrupts, exceptions, faults, traps, and abort signals to the DVMM. A GOS may be allowed, through the VMA interface, to set up a dispatch vector table, enable or disable specific event, or change the handler for specific events.

According to one embodiment, a GOS may be presented a typical interface paradigm for interrupt and exception handling. In the case of an Intel-based interface, an interrupt dispatch table (IDT) may be used to communicate between the GOS and the DVMM. In particular, an IDT allows the distributed virtual machine monitor to dispatch events of interest to a specific GOS executing on a specific virtual server. A GOS is permitted to change table entries by registering a new table or by changing entries in an existing table. To preserve isolation and security, individual vectors within the IDT may remain writeable only by the distributed virtual machine monitor, and tables and information received from the GOS are not directly writable. In one example, all interrupts and exceptions are processed initially by the distributed virtual machine monitor.

As discussed above, a virtual machine architecture (VMA) may be defined that is presented as an abstraction layer to the GOS. Any OS (e.g., Linux, Windows, Solaris, etc.) may be ported to run on a VMA in the same manner as would be performed when porting the OS to any other architecture (e.g., Alpha, Intel, MIPS, SPARC, etc.). According to one aspect of the present invention, the VMA presented to the GOS may be similar to an Intel-based architecture such as, for example, IA-32 or IA-64.

In an example VMA architecture, non-privileged instructions may be executed natively on an underlying hardware processor, without intervention. In instances when privileged registers or instructions must be accessed, the distributed virtual machine monitor may intervene. For examples, in cases where there are direct calls from the operating system, trap code in the VMA may be configured to handle these calls. In the case of exceptions (unexpected operations) such as device interrupts, instruction traps, page faults or access to a privileged instruction or register may cause an exception. In one example, the distributed virtual machine monitor may handle all exceptions, and may deliver these exceptions to the GOS via a VMA or may be handled by the VMA.

Execution Privilege Levels

Figure 7 shows an execution architecture 700 according to one aspect of the invention. In particular, architecture 700 includes a number of processor privilege levels at which various processes may be executed. In particular, there is defined a user mode level 705 having a privilege level of three (3) at which user mode programs (e.g., applications) are executed. At this level, GOS user processes 701 associated with one or more application programs are executed. Depending on the access type requested, user processes 701 may be capable of accessing one or more privilege levels as discussed further below.

There may also be a supervisor mode 706 that corresponds to a privilege level one (1) at which the GOS kernel (item 702) may be executed. In general, neither the GOS nor user processes are provided access to the physical processor directly, except when executing non-privileged instructions 709. In accordance with one embodiment, non-privileged instructions are executed directly on the hardware (e.g., a physical processor 704 within a node). This is advantageous for performance reasons, as there is less overhead processing in handling normal operating functions that may be more efficiently processed directly by hardware. By contrast, privileged instructions may be processed through the distributed virtual machine monitor (e.g., DVMM 703) prior to being serviced by any hardware. In one embodiment, only the DVMM is permitted to run at privilege level 0 (kernel mode) on the actual hardware. Virtual server isolation implies that the GOS cannot have uncontrolled access to any hardware features (such as CPU control registers) nor to certain low-level data structures (such as, for example, paging directories/tables and interrupt vectors).

In the case where the hardware is the Intel IA-32 architecture, there are four processor privilege levels. Therefore, the GOS (e.g., Linux) may execute at a level higher than kernel

mode (as the distributed virtual machine monitor, according to one embodiment, is only permitted to operate in kernel mode). In one embodiment, the GOS kernel may be executed in supervisor mode (privilege level 1) to take advantage of IA-32 memory protection hardware to prevent applications from accessing pages meant only for the GOS kernel. The GOS kernel may "call down" into the distributed virtual machine monitor to perform privileged operations (that could affect other virtual servers sharing the same hardware), but the distributed virtual machine monitor should verify that the requested operation does not compromise isolation of virtual servers. In one embodiment of the present invention, processor privilege levels may be implemented such that applications, the GOS and distributed virtual machine monitor are protected from each other as they reside in separate processor privilege levels.

Although the example shown in Figure 7 has four privilege levels, it should be appreciated that any number of privilege levels may be used. For instance, there are some architecture types that have two processor privilege levels, and in this case, the distributed virtual machine monitor may be configured to operate in the supervisor mode (privilege level (or ring) 0) and the user programs and operating system may be executed at the lower privilege level (e.g., level 1). It should be appreciated that other privilege scenarios may be used, and the invention is not limited to any particular scenario.

Example Distributed Virtual Machine Monitor Architecture

Figure 8 shows an example of a DVMM architecture according to one embodiment of the present invention. As discussed above, the DVMM is a collection of software that handles the mapping of resources from the physical realm to the virtual realm. Each hardware node (e.g., a physical processor associated with a node) executes a low-level system software that is a part of the DVMM, a microkernel, and a collection of these instances executing on a number of physical processors form a shared-resource cluster. As discussed above, each collection of cooperating (and communicating) microkernels is a distributed server. There is a one-to-one mapping of a distributed server to a distributed virtual machine monitor (DVMM). The DVMM, according to one embodiment, is as thin a layer as possible. The DVMM may be, for example, one or more software programs stored in a computer readable medium (e.g., memory, disc storage, or other medium capable of being read by a computer system).

Figure 8 shows a DVMM architecture 800 according to one embodiment of the present invention. DVMM 800 executes tasks associated with one or more instances of a virtual server (e.g., virtual server instances 801A-801B). Each of the virtual server instances store an

execution state of the server. For instance, each of the virtual servers 801A-801B store one or more virtual registers 802A-802B, respectively, that correspond to a register states within each respective virtual server.

DVMM 800 also stores, for each of the virtual servers, virtual server states (e.g., states 803A, 803B) in the form of page tables 804, a register file 806, a virtual network interface (VNIC) and virtual fiber channel (VFC) adapter. The DVMM also includes a packet scheduler 808 that schedules packets to be transmitted between virtual servers (e.g., via an InfiniBand connection or other connection, or direct process-to-process communication).

I/O scheduler 809 may provide I/O services to each of the virtual servers (e.g., through I/O requests received through the VMA). In addition, the DVMM may support its own I/O, such as communication between nodes. Each virtual device or controller includes an address that may be specified by a virtual server (e.g., in a VMA I/O request). I/O devices is abstracted as a virtual device to the virtual server (e.g., as a PCI or PCI-like device) such that the GOS may access this device. Each VIO device may be described to the GOS by a fixed-format description structure analogous to the device-independent PCI config space window.

Elements of the descriptor may include the device address, class, and/or type information that the GOS may use to associate the device with the proper driver module. The descriptor may also include, for example, one or more logical address space window definitions for device-specific data structures, analogous to memory-mapped control/status registers. The I/O scheduler 809 schedules requests received from virtual servers and distributes them to one or more I/O controllers that interface to the actual I/O hardware. More particularly, the DVMM I/O includes a set of associated drivers that moves the request onto a communication network (e.g., InfiniBand) and to an I/O device for execution. I/O may be performed to a number of devices and systems including a virtual console, CD/DVD player, network interfaces, keyboard, etc. Various embodiments of an I/O subsystem are discussed further below with respect to Figure 9.

CPU scheduler 810 may perform CPU scheduling functions for the DVMM. More particularly, the CPU scheduler may be responsible for executing the one or more GOSs executing on the distributed server. The DVMM may also include supervisor calls 811 that include protected supervisor mode calls executed by an application through the DVMM. As discussed above, protected mode instructions may be handled by the DVMM to ensure isolation and security between virtual server instances.

Packet scheduler 808 may schedule packet communication and access to actual network devices for both upper levels (e.g., GOS, applications) as well as network support within DVMM 800. In particular, packet scheduler 808 may schedule the transmission of packets on one or more physical network interfaces, and perform a mapping between virtual interfaces defined for each virtual server and actual network interfaces.

DVMM 800 further includes a cluster management component 812. Component 812 provides services and support to bind the discrete systems into a cluster and provides basic services for the microkernels within a distributed server to interact with each other. These services include cluster membership and synchronization. Component 812 includes a clustering subcomponent 813 that defines the protocols and procedures by which microkernels of the distributed servers are clustered. At the distributed server level, for example, the configuration appears as a cluster, but above the distributed server level, the configuration appears as a non-uniform memory access, multi-processor single system.

The DVMM further includes a management agent 815. This component is responsible for handling dynamic reconfiguration functions as well as reporting status and logging to other entities (e.g., a management server). Management agent 815 may receive commands for adding, deleting, and reallocating resources from virtual servers. The management agent 815 may maintain a mapping database that defines mapping of virtual resources to physical hardware.

According to various embodiments of the invention microkernels, which form parts of a DVMM, communicate with each other using Distributed Shared Memory (DSM) based on paging and/or function shipping protocols (e.g., object-level). These techniques are used to efficiently provide a universal address space for objects and their implementation methods. With this technology, the set of instances executing on the set of physical processors seamlessly and efficiently shares objects and/or pages. The set of microkernel instances may also provide an illusion of a single system to the virtual server (running on DVMM), which boots and run a single copy of a traditional operating system.

Distributed shared memory 816 is the component that implements distributed shared memory support and provides the unified view of memory to a virtual server and in turn to the Guest Operating System. DSM 816 performs memory mapping from virtual address spaces to memory locations on each of the hardware nodes. The DSM also includes a memory allocator 817 that performs allocation functions among the hardware nodes. DSM 816 also includes a coherence protocol 818 that ensures coherence in memory of the shared-memory

multiprocessor. The DSM may be, for example, a virtual memory subsystem used by the DVMM and as the foundation for the Distributed Memory Manager subsystem used by virtual servers.

DSM 816 also includes a communication subsystem that handles distributed memory communication functions. In one example, the DMM may use RDMA techniques for accessing distributed memory among a group of hardware nodes. This communication may occur, for example, over a communication network including one or more network links and switches. For instance, the cluster may be connected by a cluster interconnect layer (e.g., interconnect driver 822) that is responsible for providing the abstractions necessary to allow microkernels to communicate between nodes. This layer provides the abstractions and insulates the rest of the DVMM from any knowledge or dependencies upon specific interconnect features.

Microkernels of the DVMM communicate, for example, over an interconnect such as InfiniBand. Other types of interconnects (e.g., PCI-Express, GigaNet, Ethernet, etc.) may be used. This communication provides a basic mechanism for communicating data and control information related to a cluster. Instances of server functions performed as part of the cluster include watchdog timers, page allocation, reallocation, and sharing, I/O virtualization and other services. Examples of a software system described below transform a set of physical compute servers (nodes) having a high-speed, low latency interconnect into a partitionable set of virtual multiprocessor machines. These virtual multiprocessor machines may be any multiprocessor memory architecture type (e.g., COMA, NUMA, UMA, etc.) configured with any amount of memory or any virtual devices.

According to one embodiment, each microkernel instance of the DVMM executes on every hardware node. As discussed, the DVMM may obtain information from a management database associated with a management server (e.g., server 212). The configuration information allows the microkernel instances of the DVMM to form the distributed server. Each distributed server provides services and aggregated resources (e.g., memory) for supporting the virtual servers.

DVMM 800 may include hardware layer components 820 that include storage and network drivers 821 used to communicate with actual storage and network devices, respectively. Communication with such devices may occur over an interconnect, allowing virtual servers to share storage and network devices. Storage may be performed, for example, using FibreChannel. Networking may be performed using, for example, a physical layer

protocol such as Gigabit Ethernet. It should be appreciated that other protocols and devices may be used, and the invention is not limited to any particular protocol or device type. Layer 820 may also include an interconnect driver 822 (e.g., an InfiniBand driver) to allow individual microkernel of the DVMM running on the nodes to communicate with each other and with other devices (e.g., I/O network). DVMM 800 may also include a hardware abstraction 823 that relates virtual hardware abstractions presented to upper layers to actual hardware devices. This abstraction may be in the form of a mapping that relates virtual to physical devices for I/O, networking, and other resources.

DVMM 800 may include other facilities that perform system operations such as software timer 824 that maintains synchronization between clustered microkernel entities. Layer 820 may also include a kernel bootstrap 825 that provides software for booting the DVMM and virtual servers. Functions performed by kernel bootstrap 825 may include loading configuration parameters and the DVMM system image into nodes and booting individual virtual servers.

In another embodiment of the present invention, the DVMM 800 creates an illusion of a Virtual cache-coherent, Non-Uniform Memory Architecture (NUMA) machine to the GOS and its application. However, it should be appreciated that other memory architectures (e.g., UMA, COMA, etc.) may be used, and the invention is not limited to any particular architecture. The Virtual NUMA (or UMA, COMA, etc.) machine is preferably not implemented as a traditional virtual machine monitor, where a complete processor ISA is exposed to the guest operating system, but rather is a set of data structures that abstracts the underlying physical processors to expose a virtual processor architecture with a conceptual ISA to the guest operating system. The GOS may be ported to the virtual machine architecture in much the same way an operating system may be ported to any other physical processor architecture.

A set of Virtual Processors makes up a single virtual multiprocessor system (e.g., a Virtual NUMA machine, a Virtual COMA machine). Multiple virtual multiprocessor systems instances may be created whose execution states are separated from one another. The architecture may, according to one embodiment, support multiple virtual multiprocessor systems simultaneously running on the same distributed server.

In another example architecture, the DVMM provides a distributed hardware sharing layer via the Virtual Processor and Virtual NUMA or Virtual COMA machine. The guest operating system is ported onto the Virtual NUMA or Virtual COMA machine. This Virtual

NUMA or Virtual COMA machine provides access to the basic I/O, memory and processor abstractions. A request to access or manipulate these items is handled via APIs presented by the DVMM, and this API provides isolation between virtual servers and allows transparent sharing of the underlying hardware.

Example System Architecture

Figure 9 is a block diagram of an example system architecture upon which a virtual computing system in accordance with one embodiment of the present invention may be implemented. As discussed above, a virtual computing system may be implemented using one or more resources (e.g., nodes, storage, I/O devices, etc.) linked via an interconnect. As shown in the example system 900 in Figure 9, a system 900 may be assembled having one or more nodes 901A-901B coupled by a communication network (e.g., fabric 908). Nodes 901A-901B may include one or more processors (e.g., processors 902A-902B) one or more network interfaces (e.g., 903A-903B) through which nodes 901A-901B communicate through the network.

As discussed above, nodes may communicate through many different types of networks including, but not limited to InfiniBand and Gigabit Ethernet. More particularly, fabric 908 may include one or more communication systems 905A-905D through which nodes and other system elements communicate. These communication systems may include, for example, switches that communicate messages between attached systems or devices. In the case of a fabric 908 that implements InfiniBand switching, interfaces of nodes may be InfiniBand host channel adapters (HCAs) as are known in the art. Further, communication systems 905A-905D may include one or more InfiniBand switches.

Communication systems 905A-905D may also be connected by one or more links. It should be appreciated, however, that other communication types (e.g., Gigabit Ethernet) may be used, and the invention is not limited to any particular communication type. Further, the arrangement of communication systems as shown in Figure 9 is merely an example, and a system according to one embodiment of the invention may include any number of components connected by any number of links in any arrangement.

Node 901A may include local memory 904 which may correspond to, for example, the node physical memory map 601 shown in Figure 6. More particularly, a portion of memory 904 may be allocated to a distributed shared memory subsystem which can be used for supporting virtual server processes.

Data may be stored using one or more storage systems 913A-913B or 920, 921 and 922. These storage systems may be, for example, network attach storage (NAS) or a storage area network (SAN) as are well-known in the art. Such storage systems may include one or more interfaces (e.g., interface 918) that are used to communicate data between other system elements. Storage system may include one or more components including one or more storage devices (e.g., disks 914), one or more controllers (e.g., controllers 915, 919), one or more processors (e.g., processor 916), memory devices (e.g., device 917), or interfaces (e.g., interface 918). Such storage systems may implement any number of communication types or protocols including Fibre Channel, SCSI, Ethernet, or other communication types.

Storage systems 913 may be coupled to fabric 908 through one or more interfaces. In the case of a fabric 908 having an InfiniBand switch architecture; such interfaces may include one or more target channel adaptors (TCAs) as are well-known in the art. System 900 may include one or more I/O systems 906A-906B. These I/O systems 906A-906B may include one or more I/O modules 912 that perform one or more I/O functions on behalf of one or more nodes (e.g., nodes 901A-901B). In one embodiment, an I/O system (e.g., system 906A) includes a communication system (e.g., system 911) that allows communication between one or more I/O modules and other system entities. In one embodiment, communication system 911 includes an InfiniBand switch.

Communication system 911 may be coupled to one or more communication systems through one or more links. Communication system 911 may be coupled in turn to I/O modules via one or more interfaces (e.g., target channel adapters in the case of InfiniBand). I/O modules 912 may be coupled to one or more other components including a SCSI network 920, other communication networks (e.g., network 921) such as, for example, Ethernet, a FibreChannel device or network 922.

For instance, one or more storage systems (e.g., systems 913) or storage networks may be coupled to a fabric though an I/O system. In particular, such systems or networks may be coupled to an I/O module of the I/O system, such as by a port (e.g., SCSI, FibreChannel, Ethernet, etc.) of an I/O module coupled to the systems or networks. It should be appreciated that systems, networks or other elements may be coupled to the virtual computing system in any manner (e.g., coupled directly to the fabric, routed through other communication devices or I/O systems), and the invention is not limited to the number, type, or placement of connections to the virtual computing system.

Modules 912 may be coupled to other devices that may be used by virtual computing systems such as a graphics output 923 that may be coupled to a video monitor, or other video output 924. Other I/O modules may perform any number of tasks and may include any number and type of interfaces. Such I/O systems 906A-906B may support, for virtual servers of a virtual computing system, I/O functions requested by a distributed virtual machine monitor in support of the GOS in its applications.

As discussed above, I/O requests may be sent to I/O controllers (e.g., I/O modules 912) over multiple communication paths within fabric 908. The I/O modules 912 service the requests by routing the requests to virtual or actual hardware that performs the I/O request, and returns status or other information to the distributed virtual machine monitor.

According to one embodiment, GOS I/O devices are virtualized devices. For example, virtual consoles, virtual block devices, virtual SCSI, virtual Host Bus Adapters (HBAs) and virtual network interface controllers (NICs) may be defined which are serviced by one or more underlying devices. Drivers for virtual I/O devices may be multi-path in that the requests may be sent over one or more parallel paths and serviced by one or more I/O modules. These multi-path drivers may exist within the GOS, and may be serviced by drivers within the DVMM. Further, these multi-path requests may be serviced in parallel by parallel-operating DVMM drivers which initiate parallel (multi-initiate) requests on hardware.

In one embodiment, virtual NICs may be defined for a virtual server that allow multiple requests to be transferred from a node (e.g., node 901A) through a fabric 908 to one or more I/O modules 912. Such communications may occur in parallel (e.g., over parallel connections or networks) and may occur, for instance, over full duplex connections. Similarly, a virtual host bus adapter (HBA) may be defined that can communicate with one or more storage systems for performing storage operations. Requests may be transmitted in a multi-path manner to multiple destinations. Once received at one or more destinations, the parallel requests may be serviced (e.g., also in parallel). One example virtual storage architecture is discussed below with respect to Figure 10.

System 900 may also be connected to one or more other communication networks 909 or fabrics 910, or a combination thereof. In particular, system 900 may connect to one or more networks 909 or fabrics 910 through a network communication system 907. In one embodiment, network communication system 907 may be switch, router or other device that translates information from fabric 908 to outside entities such as hosts, networks, nodes or other systems or devices.

Virtual Storage Adapter Architecture

Figure 10 is a block diagram of an example system architecture for a virtual storage system according to one embodiment of the present invention. As discussed above, a virtual computing system may implement a virtual storage adapter architecture wherein actual storage interfaces are virtualized and presented to an operating system (e.g., a GOS) and its applications. According to one embodiment of the present invention, a virtual storage adapter may be defined that is supported by one or more physical hardware (e.g., FibreChannel (FC) adapter (HBA), IB Fabric) and/or software (e.g., high-availability logic) resources. Because such an adapter is virtualized, details of the underlying software and hardware may be hidden from the operating system and its associated software applications.

More particularly, a virtual storage adapter (e.g., an HBA) may be defined that is supported by multiple storage resources, the storage resources being capable of being accessed over multiple data paths. According to one aspect of the present invention, the fact that there are more than one resource (e.g., disks, paths, etc.) that are used to support the virtual adapter may be hidden from the operating system. To accomplish this abstraction of underlying resources, the operating system may be presented a virtualized adapter interface that can be used to access the underlying resources transparently. Such access may be accomplished, for example, using the I/O and multipath access methods discussed above.

As discussed, a virtual adapter abstraction may be implemented in traditional multi-node, cluster or grid computing systems as are known in the art. Alternatively, a virtual adapter abstraction may be implemented in single node systems (e.g., having one or more processors) or may be implemented in a virtual computing system as discussed above. In any case, underlying software and/or hardware resources may be hidden from the operating system (e.g., a GOS in the case of the virtual computing system examples described above). However, it should be appreciated that a virtual storage adapter architecture may be used with any type of computing system, and that the invention is not limited to any particular computing architecture type.

Figure 10 shows a particular example of storage architecture 1000 that may be used with a virtual computing system according to various embodiments of the present invention. More specifically, one or more nodes 1001A-1001Z supporting a virtual server (VS) may access a virtual adapter according to one embodiment of the invention to perform storage operations. As discussed above, tasks executing on a node may access a virtual device (e.g. a virtual storage adapter) using a virtual interface associated with the virtual device. The

interface may be presented by, for example, software drivers as discussed above. According to one embodiment, these software drivers do not provide direct hardware contact, but provide support for a particular set of devices (e.g., storage). These drivers may include upper level and lower level drivers as discussed above with respect to I/O functions. A Distributed Virtual Machine Monitor (DVMM) I/O layer may receive requests for access to the virtual device (e.g., virtual storage adapter) from lower level drivers and process the requests as necessary. For instance, the DVMM I/O layer translates requests for access to a virtual storage adapter and sends the translated requests to one or more I/O systems (e.g., system 1003) for processing.

As discussed, processors of a node (e.g., processor 1005) may initiate and complete I/O operations concurrently. Processors may also be permitted to transmit requests over multiple paths to a destination storage device to be serviced. For instance, node 1001A may send multiple requests from one or more interfaces 1006 through a communication network (e.g., fabric 1002) to an I/O system 1003 for processing. System 1003 may include one or more interfaces and I/O processing modules (collectively 1007) for servicing I/O requests. These I/O requests may be storage requests directed to a storage device coupled to I/O system 1003. For example, I/O system may serve as a gateway to a FibreChannel (1011) or other type of storage network (1012). Parallel requests may be received at a destination device, and serviced. Responses may also be sent over parallel paths for redundancy or performance reasons. Further, fabric 1002 may have any number of storage entities (1013) coupled to fabric 1002, including one or more storage systems or storage networks. Such storage entities may be directly attached to fabric 1002 or be coupled indirectly by one or more communication devices and/or networks.

According to one embodiment of the present invention, the virtual adapter (e.g., a virtual HBA or VHBA) may be defined for a particular virtual server (VS). The virtual adapter is assigned a virtual identifier through which storage resources are referenced and accessed. In one embodiment, the virtual identifier is a World Wide Node Name (WWNN) that uniquely identifies a VHBA. For instance, a virtual HBA may defined in the virtual computing system as "VHBA-1" or some other identifier having a WWNN address of 01-08-23-09-10-35-20-18, for example, or other valid WWWN identifier. In one example, virtual WWNN identifiers are provided by a software vendor providing virtualization system software. It should be appreciated, however, that any other identifier used to identify storage may be used, and that the invention is not limited to WWNN identifiers.

VHBAs having WWNN identifiers may be assigned to virtual servers (VSs), for example, using an interface of a management program. For instance, a user or program may present an interface through which one or more VHBAs may be assigned to a particular VS. Because WWNN indentifiers must be globally unique within a system, the identifiers may be administered centrally by a management server (e.g., manager 1004). In one embodiment, the management server maintains a database 1008 of available WWNN identifiers that may be used by the virtual computing system. These WWNN identifiers may be associated with corresponding virtual adapters defined in the virtual computing system, and allocated to virtual servers. Manager 1004 may communicate with one or more components of the virtual computing system through one or more links. For instance, manager 1004 may be coupled to fabric 1002 and may communicate using one or more communication protocols. Further, manager 1004 may be coupled to the virtual computing system through a data communication network 1013. More particularly, manager 1004 may be coupled to fabric 1002 through a data communication network 1013 through I/O system 1014. It should be appreciated that manager 1004 may be coupled to the virtual communication system in any manner, and may communicate using any protocol.

In one embodiment, a particular VHBA has only one WWNN assigned. This is beneficial, as mappings to underlying resources may change, yet the VHBA (and its assigned WWNN) do not change. A user (e.g., an administrator) may assign an available WWNN to the VHBA using a management interface associated with a management server (e.g., manager 1004).

Also, within the management interface, the user may be permitted to associate storage entities with one or more VHBAs. For instance, SCSI Target/LUNs may be associated with a VHBA. The Target (or Target ID) represents a hardware entity attached to a SCSI FC interconnect. Storage entities, referred to by a Logical Unit Number (LUN), may be mapped to a VHBA which then permits the VS associated with the VHBA to access a particular LUN. Such mapping information may be maintained, for example, in a database by the management server. It should be appreciated that any storage element may be associated with a virtual adapter, and that the invention is not limited to any number or particular type of storage element or identification/addressing convention.

In support of multi-pathing to various storage entities, there may be one or more options by which data is multi-pathed. For example, associated with each storage entity may be path preference (e.g., path affinity) information that identifies a preferred path among a

number of available paths. For example, if the number of outstanding I/O requests becomes excessive, or if a path fails, an alternate path may be used. Another option may include a load balancing feature that allows an I/O server to distribute I/O among one or more gateway ports to a storage entity. For instance, an I/O server may attempt to distribute requests (or data traffic) equally among a number of gateway ports. Further, an I/O server having multiple gateway ports to a particular destination entity may allow gateway port failover in the case where a primary gateway port fails.

According to one embodiment, each of these multi-pathing features are transparent to the GOS and its applications. That is, multi-pathing configuration and support (and drivers) need not exist within the GOS. Yet, according to one embodiment of the present invention, because multi-pathing is performed at lower levels, the GOS is provided the performance and reliability benefits of multi-pathing without the necessity of exposing underlying support structures of multi-pathing hardware and software. Such a feature is beneficial, particularly for operating systems and applications that do not support multi-pathing.

Conclusion

In summary, a virtual storage adapter architecture is provided. This virtual storage adapter architecture allows, for example, redundancy, multi-pathing features, and underlying hardware changes without the necessity of changes in the application or operating system that uses the virtual storage adapter architecture. Such virtual storage adapter architecture may be used, for example, in single-node or multi-node computer systems (e.g., grid-based, cluster-based, etc.). Further, such virtual storage adapter architecture may be used in a virtual computing system that executes on one or more nodes.

In one such virtual computing system as discussed above that executes on one or more nodes, a level of abstraction is created between a set of physical processors among the nodes and a set of virtual multiprocessor partitions to form a virtualized data center. This virtualized data center comprises a set of virtual, isolated systems separated by boundaries. Each of these systems appears as a unique, independent virtual multiprocessor computer capable of running a traditional operating system and its applications. In one embodiment, the system implements this multi-layered abstraction via a group of microkernels that are a part of a distributed virtual machine monitor (DVMM) to form a distributed server, where each of the microkernels communicates with one or more peer microkernel over a high-speed, low-latency interconnect.

Functionally, a virtual data center is provided, including the ability to take a collection of servers and execute a collection of business applications over the compute fabric. Processor, memory and I/O are virtualized across this fabric, providing a single system image, scalability and manageability. According to one embodiment, this virtualization is transparent to the application.

Ease of programming and transparency is achieved by supporting a shared memory programming paradigm. Both single and multi-threaded applications can be executed without modification on top of various embodiments of the architecture.

According to one embodiment, a part of the distributed virtual machine monitor (DVMM), a microkernel, executes on each physical node. A set of physical nodes may be clustered to form a multi-node distributed server. Each distributed server has a unique memory address space that spans the nodes comprising it. A cluster of microkernels form a distributed server which exports a VMA interface. Each instance of this interface is referred to as a virtual server.

Because there is isolation between the operating system and its application from the underlying hardware, the architecture is capable of being reconfigured. In one embodiment, capability for dynamically reconfiguring resources is provided such that resources may be allocated (or deallocated) transparently to the applications. In particular, capability may be provided to perform changes in a virtual server configuration (e.g., node eviction from or integration to a virtual processor or set of virtual processors). In another embodiment, individual virtual processors and partitions can span physical nodes having one or more processors. In one embodiment, physical nodes can migrate between virtual multiprocessor systems. That is, physical nodes can migrate across distributed server boundaries.

According to another embodiment of the invention, copies of a traditional multiprocessor operating system boot into multiple virtual servers. According to another embodiment of the invention, virtual processors may present an interface to the traditional operating system that looks like a pure hardware emulation or the interface may be a hybrid software/hardware emulation interface.

It should be appreciated that the invention is not limited to each of embodiments listed above and described herein, but rather, various embodiments of the invention may be practiced alone or in combination with other embodiments.

Having thus described several aspects of at least one embodiment of this invention, it is to be appreciated that various alterations, modifications and improvements will readily occur to

those skilled in the art. Such alterations, modifications, and improvements are intended to be part of this disclosure, and are intended to be within the spirit and scope of the invention. Accordingly, the foregoing description is by way of example only.

CLAIMS

1. A computer comprising:

one or more storage entities, at least one of which is capable of servicing one or more requests for access to the one or more storage entities;

one or more physical storage adapters used to communicate the one or more requests for access to the one or more storage entities; and

a virtual storage adapter adapted to receive the one or more requests and adapted to forward the one or more requests to the one or more physical storage adapters.

2. The computer system according to claim 1, wherein the virtual storage adapter is associated with a virtual server in a virtual computing system.

3. The computer system according to claim 1, wherein the computer system includes a multi-node computer system, at least two nodes of which are adapted to access the virtual storage adapter.

4. The computer system according to claim 1, wherein the virtual storage adapter is identified by a globally unique identifier.

5. The computer system according to claim 4, wherein the unique identifier includes a World Wide Node Name (WWNN) identifier.

6. The computer system according to claim 1, wherein the virtual storage adapter is a virtual host bus adapter (HBA).

7. The computer system according to claim 1, further comprising a plurality of communication paths coupling a processor of the computer system and at least one of the one or more storage entities, the virtual storage adapter being capable of directing the one or more requests over the plurality of communication paths.

8. The computer system according to claim 7, wherein at least one of the one or more requests is translated to multiple request messages being transmitted in parallel over the plurality of communication paths.

9. The computer system according to claim 7, wherein at least one of the plurality of communication paths traverses a switched communication network.

10. The computer system according to claim 9, wherein the switched communication network includes an InfiniBand switched fabric.

11. The computer system according to claim 9, wherein the switched communication network includes a packet-based network.

12. The computer system according to claim 1, further comprising a virtualization layer that maps the virtual storage adapter to the one or more physical storage adapters.

13. The computer system according to claim 12, further comprising a plurality of processors and wherein the virtualization layer is adapted to define one or more virtual servers, at least one of which presents a single computer system interface to an operating system.

14. The computer system according to claim 13, wherein the single computer system interface defines a plurality of instructions, and wherein at least one of the plurality of instructions is directly executed on at least one of the plurality of processors, and at least one other of the plurality of instructions is handled by the virtualization layer.

15. The computer system according to claim 1, further comprising a plurality of processors, wherein each of the plurality of processors executes a respective instance of a microkernel program, and wherein each of the respective instances of the microkernel program are adapted to communication to cooperatively share access to storage via the virtual storage adapter.

16. The computer system according to claim 13, wherein the virtual storage adapter is associated with the one or more virtual servers.

17. The computer system according to claim 4, further comprising a manager adapted to assign the unique identifier to the virtual storage adapter.

18. The computer system according to claim 16, wherein a change in at least one of the one or more physical storage adapters is transparent to the operating system.

19. The computer system according to claim 16, further comprising configuration information identifying a storage configuration, and wherein a change in at least one of the one or more physical storage adapters is transparent to the operating system.

20. The computer system according to claim 8, further comprising at least one I/O server, wherein the parallel access requests are serviced in parallel by the I/O server.

21. The computer system according to claim 8, wherein the at least one of the one or more storage entities receives the multiple request messages and services the multiple request messages in parallel.

22. The computer system according to claim 1, wherein the virtual storage adapter is associated with a node in a multi-node computing system.

23. The computer system according to claim 22, wherein the multi-node computing system is a grid-based computing system.

24. The computer system according to claim 22, wherein the multi-node computing system is a cluster-based computing system.

25. The computer system according to claim 1, wherein the virtual storage adapter is associated with a single computer system.

26. The computer system according to claim 22, wherein the multi-node computing system supports a virtual computing system that executes on the multi-node computing system, and wherein the virtual computing system is adapted to access the virtual storage adapter.

27. The computer system according to claim 25, wherein the single computer system supports a virtual computing system that executes on the single computer system, and wherein the virtual computing system is adapted to access the virtual storage adapter.

28. The computer system according to claim 22, wherein the virtual storage adapter is identified by a globally unique identifier.

29. The computer system according to claim 28, wherein the globally unique identifier includes a World Wide Node Name (WWNN) identifier.

30. The computer system according to claim 25, wherein the virtual storage adapter is identified by a globally unique identifier.

31. The computer system according to claim 30, wherein the globally unique identifier includes a World Wide Node Name (WWNN) identifier.

32. A computer-implemented method in a computer system having one or more storage entities, at least one of which is capable of servicing one or more requests for access to the one or more storage entities, and having one or more physical storage adapters used to communicate the one or more requests for access to the one or more storage entities, the method comprising an act of:

providing for a virtual storage adapter, the virtual adapter adapted to perform acts of:

receiving the one or more requests; and

forwarding the one or more requests to the one or more physical storage adapters.

33. The method according to claim 32, further comprising an act of associating the virtual storage adapter with a virtual server in a virtual computing system.

34. The method according to claim 32, wherein the computer system includes a multi-node computer system, and wherein at least two nodes of the computer system each perform an act of accessing the virtual storage adapter.

35. The method according to claim 32, further comprising an act of identifying the virtual storage adapter by a globally unique identifier.

36. The method according to claim 35, wherein the act of identifying the virtual storage adapter includes an act of identifying the virtual storage adapter by a World Wide Node Name (WWNN) identifier.

37. The method according to claim 32, wherein the act of providing for a virtual storage adapter includes an act of providing a virtual host bus adapter (HBA).

38. The method according to claim 32, wherein the computer system further comprises a plurality of communication paths coupling a processor of the computer system and at least one of the one or more storage entities, and wherein the method further comprises an act of directing, by the virtual storage adapter, the request over the plurality of communication paths.

39. The method according to claim 32, wherein the computer system further comprises a plurality of communication paths coupling a processor of the computer system and at least one of the one or more storage entities, and wherein the method further comprising acts of translating at least one of the one or more requests to multiple request messages and transmitting the multiple request messages in parallel over the plurality of communication paths.

40. The method according to claim 38, wherein at least one of the plurality of communication paths traverses a switched communication network.

41. The method according to claim 40, wherein the switched communication network includes an InfiniBand switched fabric.

42. The method according to claim 40, wherein the switched communication network includes a packet-based network.

43. The method according to claim 32, further comprising an act of mapping the virtual storage adapter to the one or more physical storage adapters.

44. The method according to claim 43, wherein the act of mapping is performed in a virtualization layer of the computer system.

45. The method according to claim 44, wherein the computer system further comprises a plurality of processors, and wherein the method further comprises an act of defining one or more virtual servers, at least one of which presents a single computer system interface to an operating system.

46. The method according to claim 44, wherein the computer system further comprises a plurality of processors, and wherein the method further comprises an act of defining one or more virtual servers, at least one of which presents a single computer system interface to an operating system.

47. The method according to claim 45, wherein the act of defining is performed by the virtualization layer.

48. The method according to claim 46, wherein the act of defining is performed by the virtualization layer.

49. The method according to claim 46, wherein the single computer system interface defines a plurality of instructions, and wherein the method further comprises an act of executing at least one of the plurality of instructions directly on at least one of the plurality of processors, and handling, by the virtualization layer, at least one other of the plurality of instructions.

50. The method according to claim 32, wherein the computer system comprises a plurality of processors, and wherein each of the plurality of processors performs an act of executing a respective instance of a microkernel program, and wherein each of the respective instances of the microkernel program communicate to cooperatively share access to storage via the virtual storage adapter.

51. The method according to claim 46, further comprising an act of associating the virtual storage adapter with the one or more virtual servers.

52. The method according to claim 35, wherein the computer system further comprises a manager, and wherein the method further comprises an act of assigning, by the manager, the unique identifier to the virtual storage adapter.

53. The method according to claim 45, wherein a change in at least one of the one or more physical storage adapters is transparent to the operating system.

54. The method according to claim 45, further comprising an act of maintaining configuration information identifying a storage configuration, and wherein a change in at least one of the one or more physical storage adapters is transparent to the storage configuration.

55. The method according to claim 39, wherein the computer system further comprises at least one I/O server, wherein the parallel access request messages are serviced in parallel by the I/O server.

56. The method according to claim 39, further comprising acts of receiving, by the at least one of the one or more storage entities, the multiple request messages, and servicing the multiple request messages in parallel.

57. The method according to claim 32, further comprising an act of associating the virtual storage adapter with a node in a multi-node computing system.

58. The method according to claim 57, wherein the multi-node computing system is a grid-based computing system.

59. The method according to claim 57, wherein the multi-node computing system is a cluster-based computing system.

60. The method according to claim 32, further comprising an act of associating the virtual storage adapter with a single computer system.

61. The method according to claim 57, wherein the multi-node computing system supports a virtual computing system that executes on the multi-node computing system, and wherein the

method further comprises an act of accessing, by the virtual computing system, the virtual storage adapter.

62. The method according to claim 60, wherein the single computer system supports a virtual computing system that executes on the single computer system, and wherein the method further comprises an act of accessing, by the virtual computing system, the virtual storage adapter.

63. The method according to claim 57, further comprising an act of identifying the virtual storage adapter by a globally unique identifier.

64. The method according to claim 63, wherein the act of identifying the virtual storage adapter includes an act of identifying the virtual storage adapter by a World Wide Node Name (WWNN) identifier.

65. The method according to claim 60, further comprising an act of identifying the virtual storage adapter by a globally unique identifier.

66. The method according to claim 65, wherein the globally unique identifier includes a World Wide Node Name (WWNN) identifier.

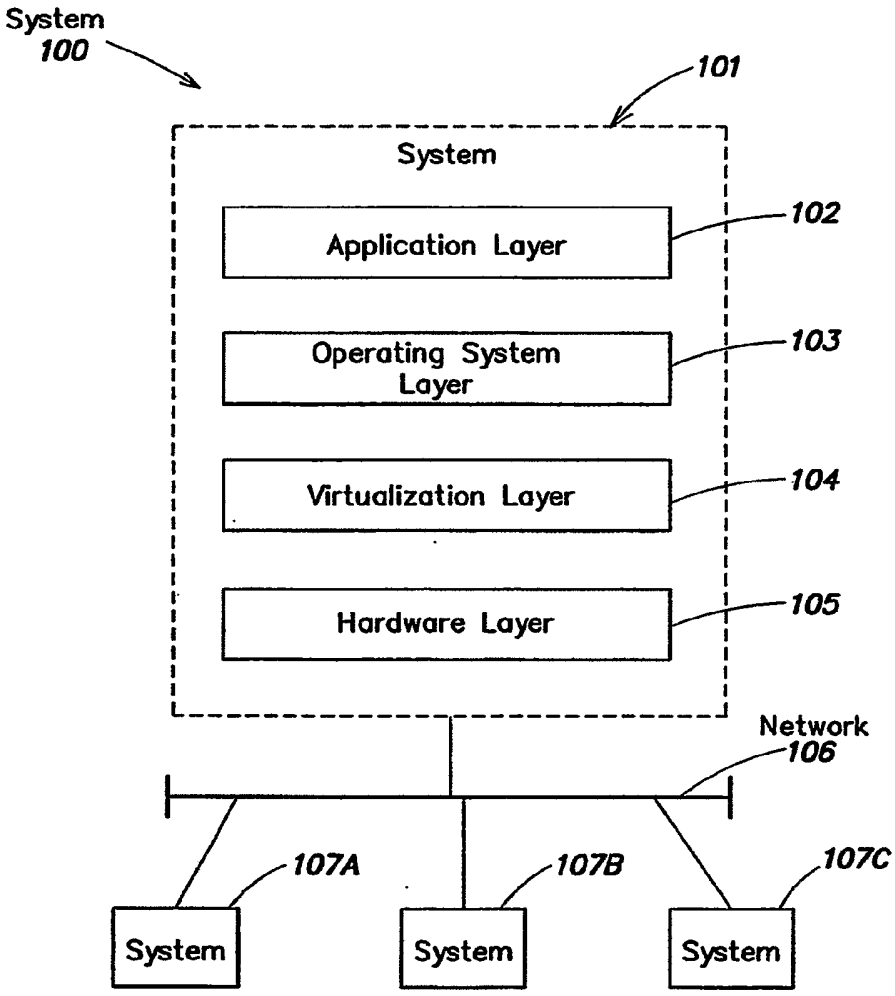


FIG. 1

CONFIRMATION

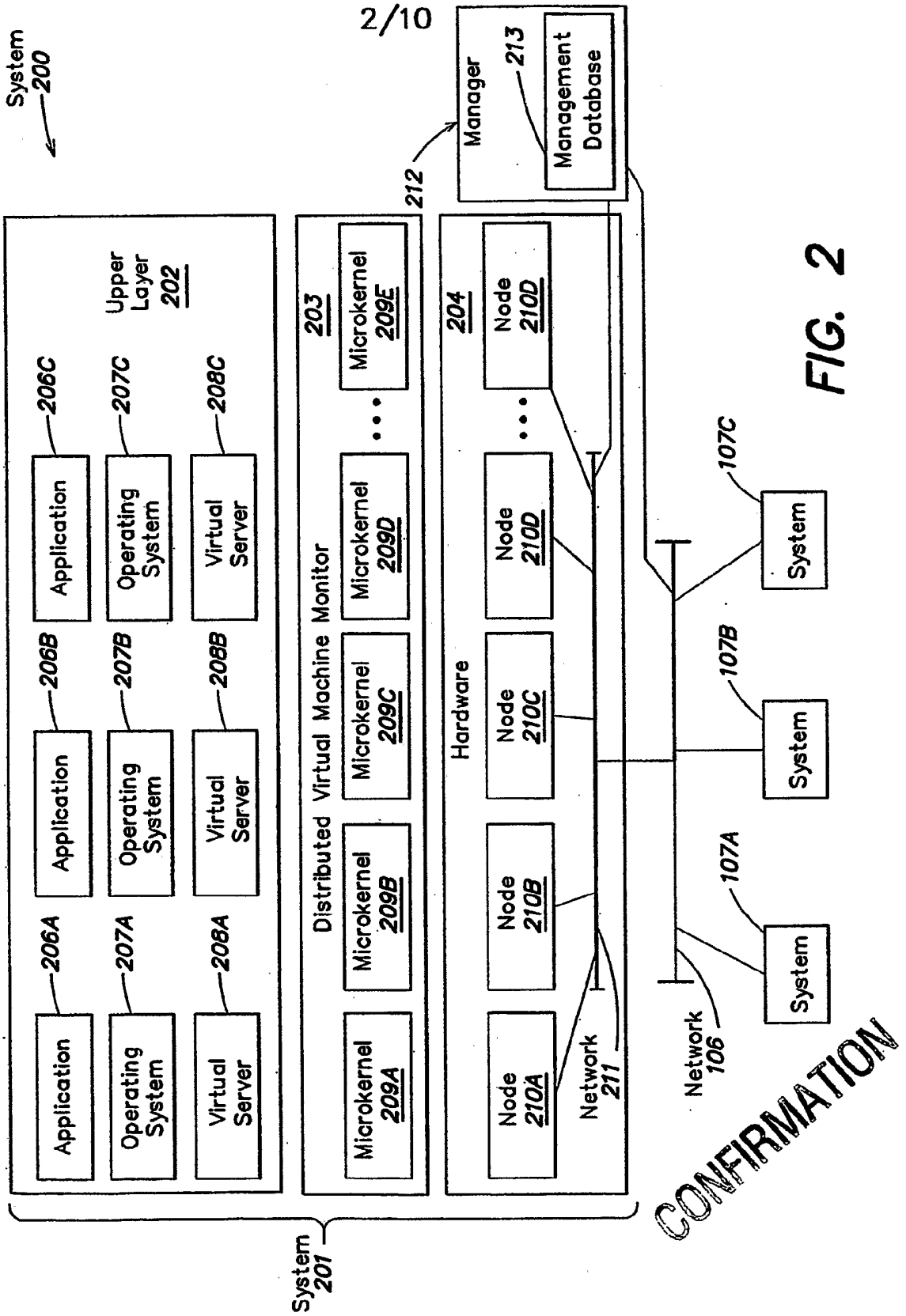

System 200
Upper Layer 202
Application 206A
Operating System 207A
Virtual Server 208A
Application 206B
Operating System 207B
Virtual Server 208B
Application 206C
Operating System 207C
Virtual Server 208C
Distributed Virtual Machine Monitor 203
Microkernel 209A
Microkernel 209B
Microkernel 209C
Microkernel 209D
Microkernel 209E
Hardware 204
Node 210A
Node 210B
Node 210C
Node 210D
Node 210D
Manager 212
Management Database 213
Network 211
Network 106
System 107A
System 107B
System 107C
System 201
CONFIRMATION

FIG. 2

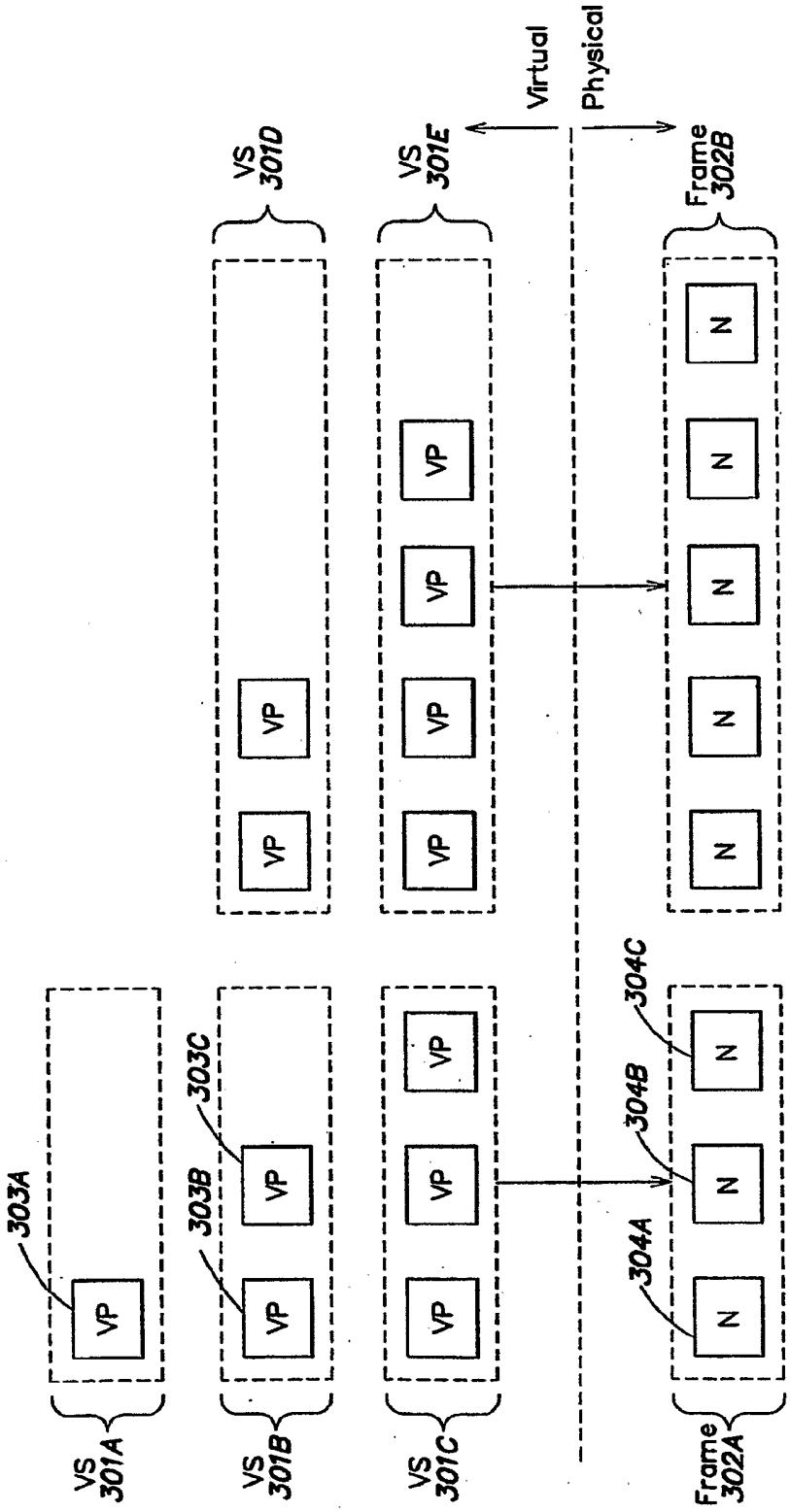


FIG. 3

CONFIRMATION

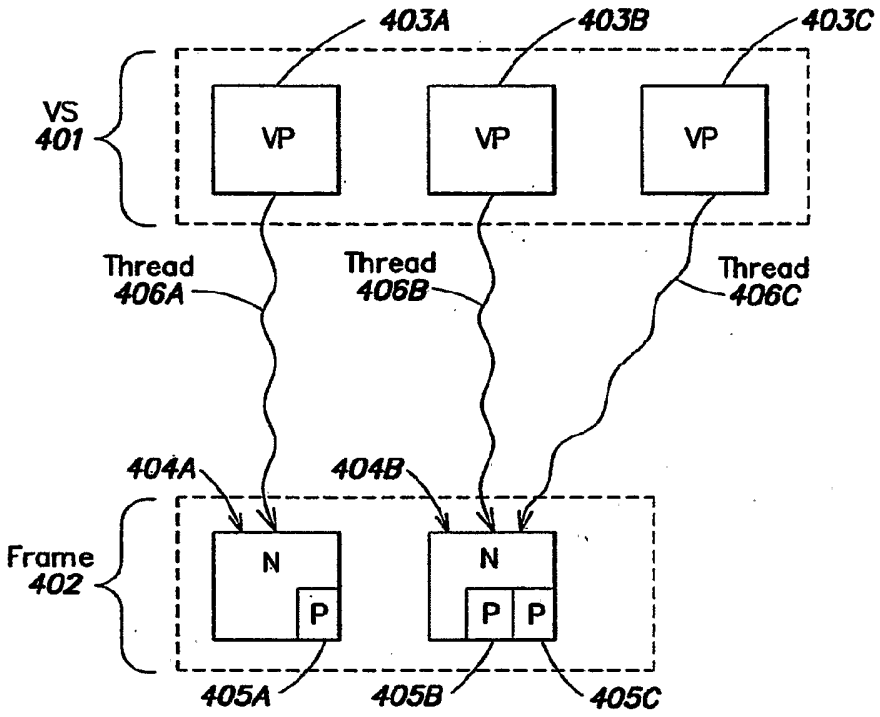


FIG. 4

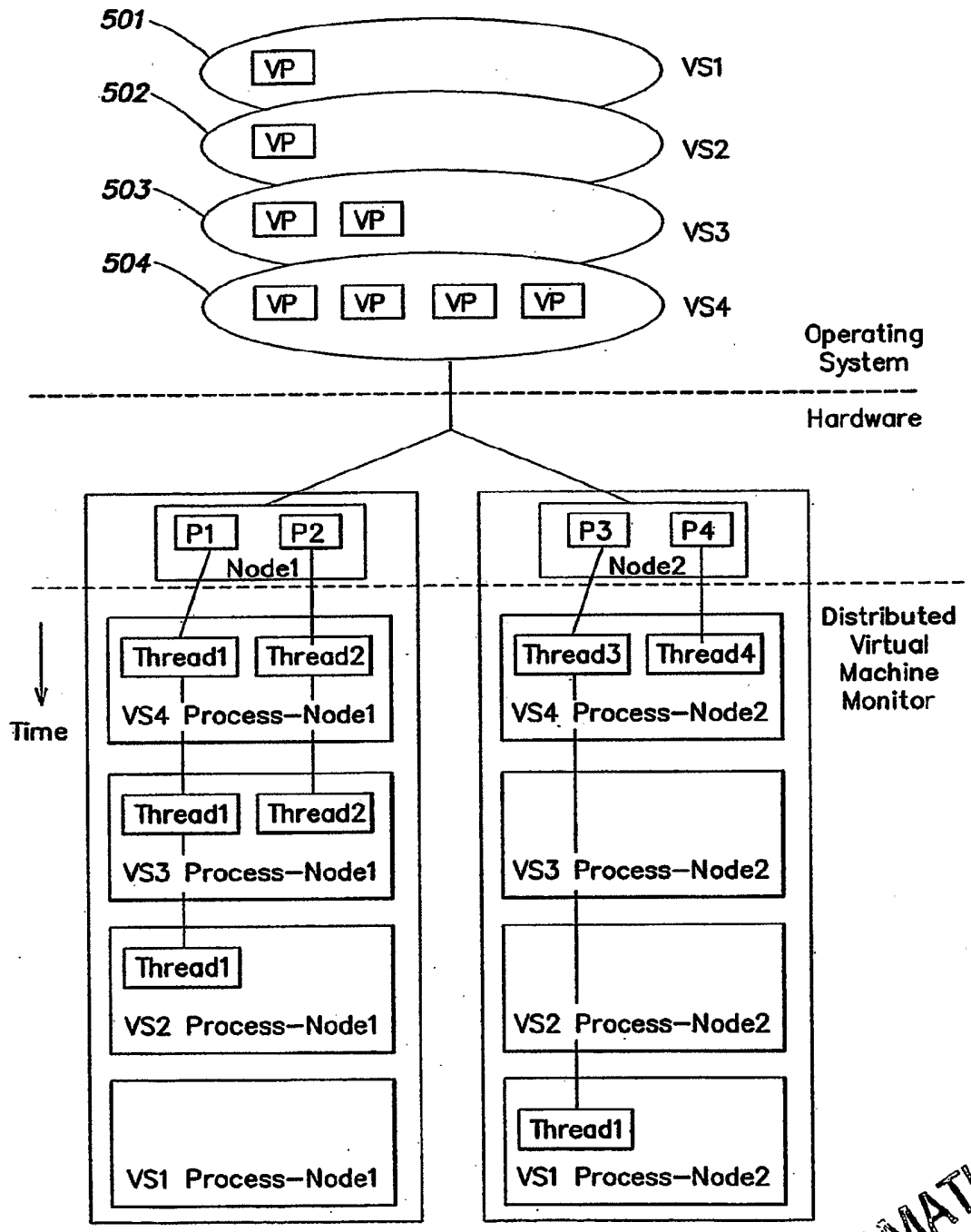


FIG. 5

CONFIRMATION

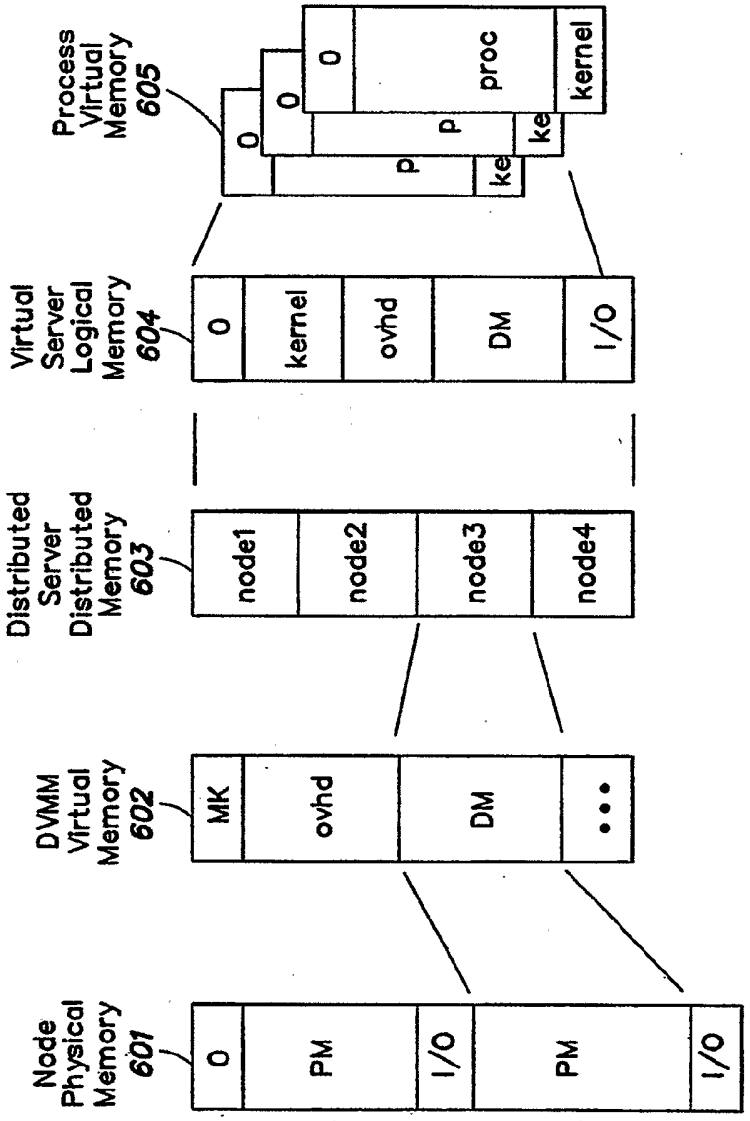


FIG. 6

CONFIRMATION

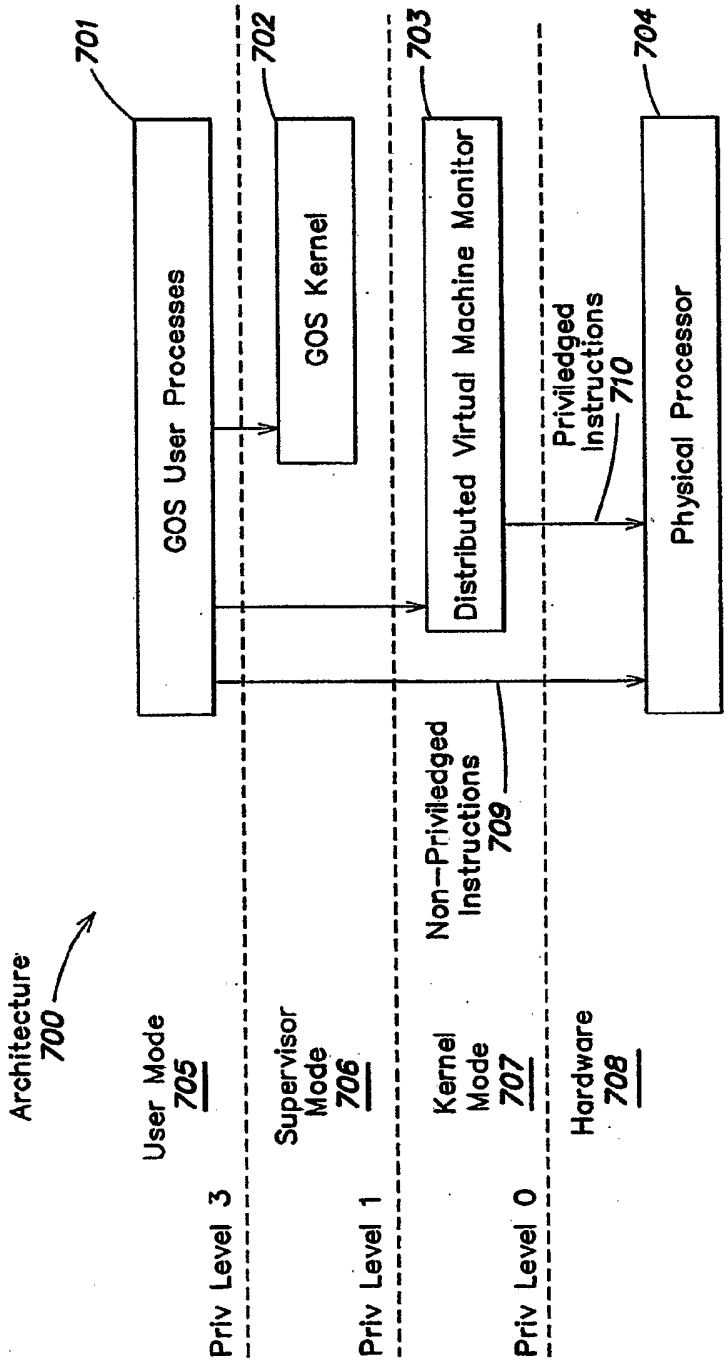


FIG. 7

CONFIRMATION

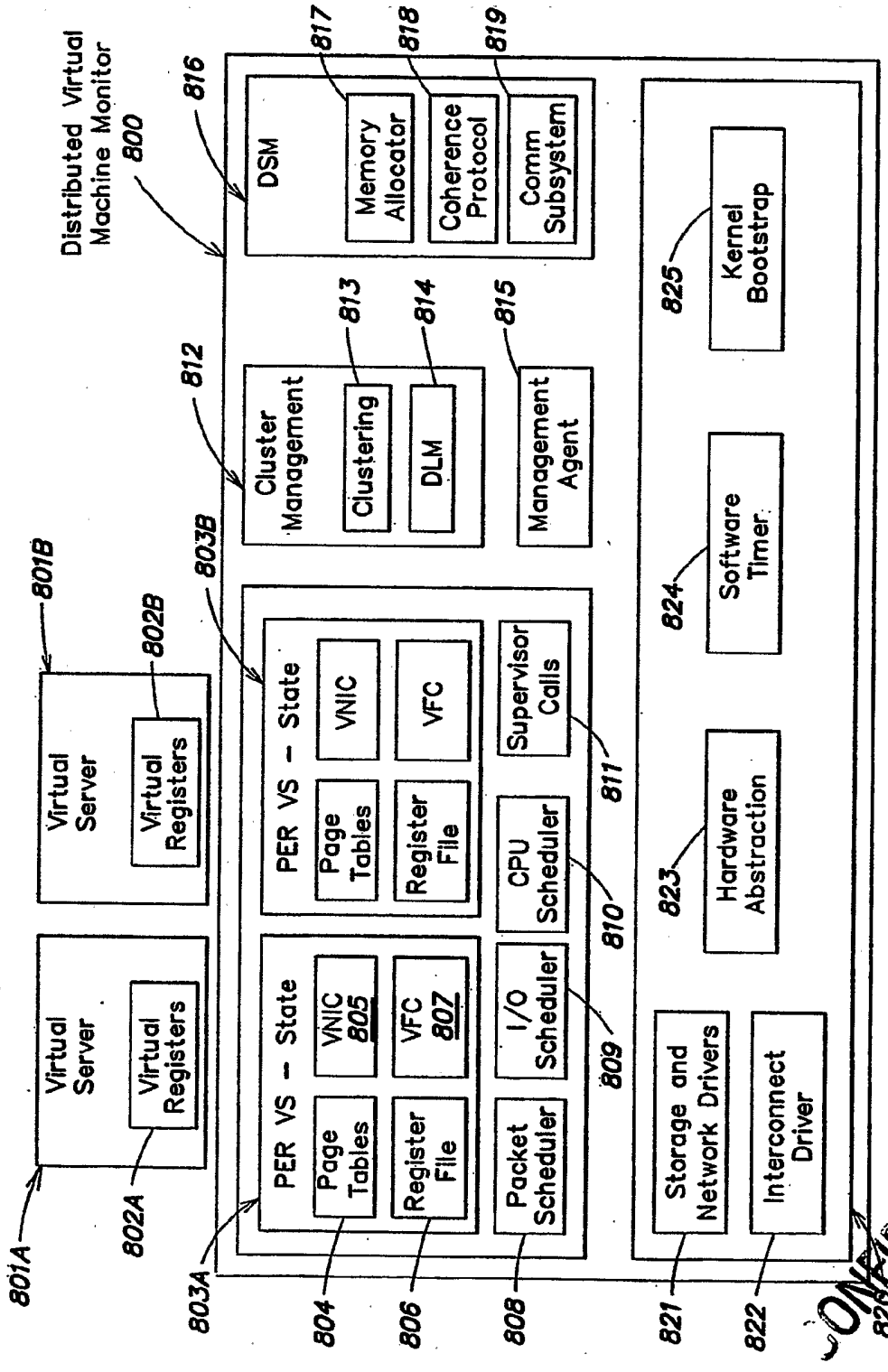


FIG. 8

CONFIRMATION

RECTIFIED SHEET (RULE 91)
ISA/EP

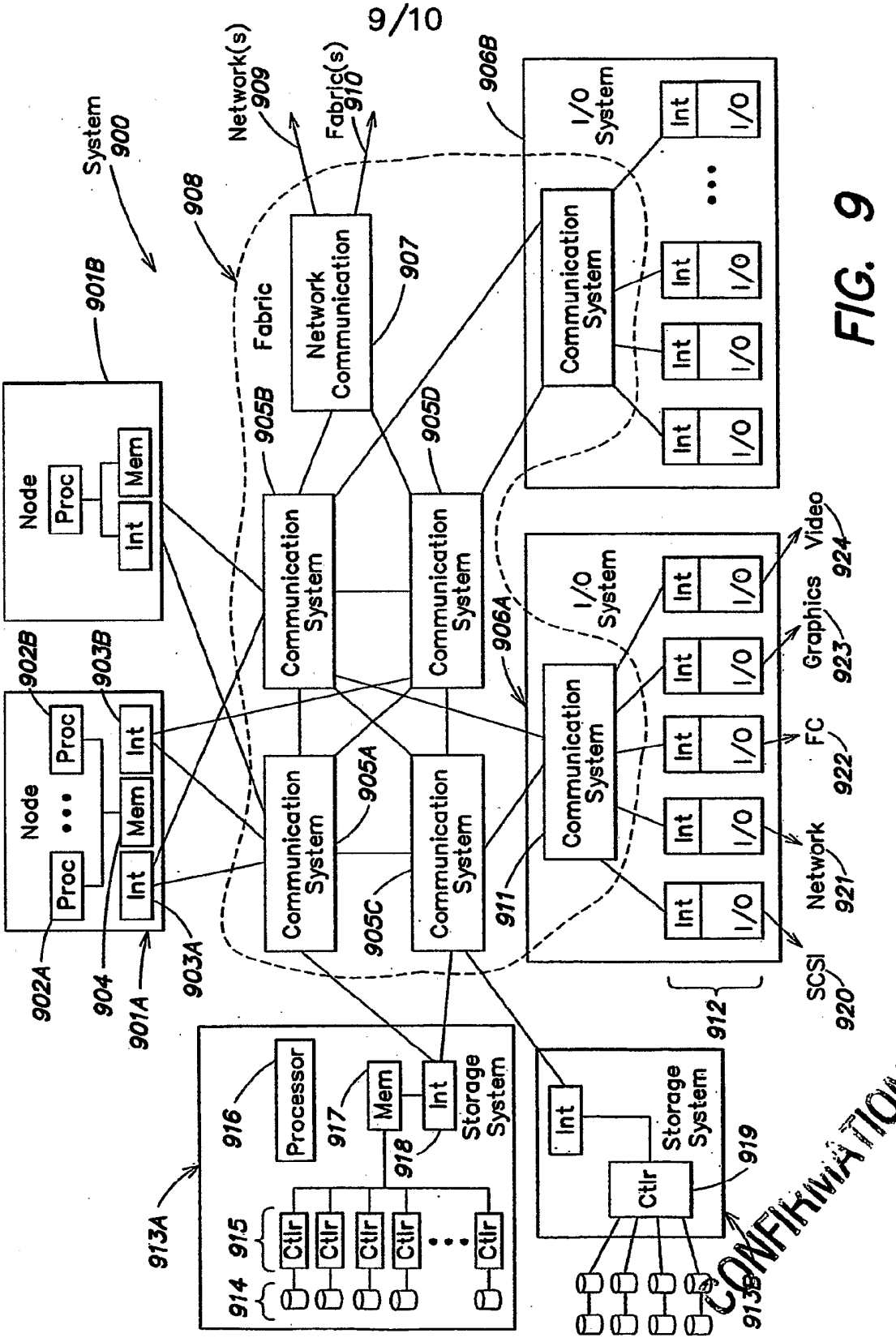


FIG. 9

CONFIRMATION

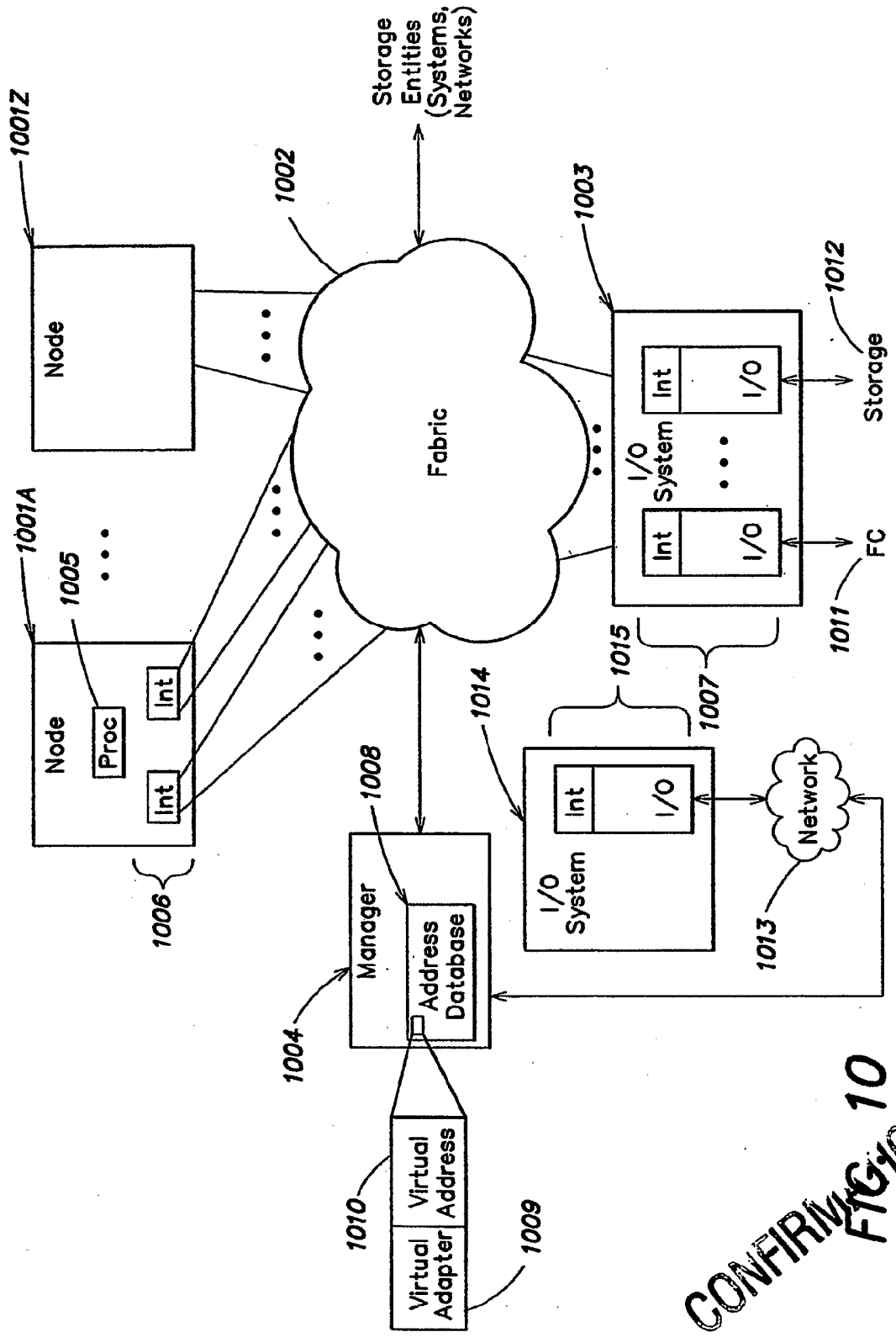


FIG. 10

CONFIRMATION COPY