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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
9 November 2006 (09.11.2006)

PCT

(10) International Publication Number
WO 2006/118926 A2

(51) International Patent Classification:
*G06F 3/06* (2006.01)

(21) International Application Number:
PCT/US2006/015917

(22) International Filing Date: 27 April 2006 (27.04.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
11/119,118 29 April 2005 (29.04.2005) US

(71) Applicant *(for all designated States except US)*: **NETWORK APPLIANCE, INC.** [US/US]; 495 East Java Drive, Sunnyvale, California 94089 (US).

(72) Inventor: **JERNIGAN, Richard, P., IV**; 1016 Rice Avenue, Ambridge, Pennsylvania 15003 (US).

(74) Agent: **BARBAS, Charles, J.**; CESARI AND MCKENNA, LLP, 88 Black Falcon Avenue, Boston, Massachusetts 02210 (US).

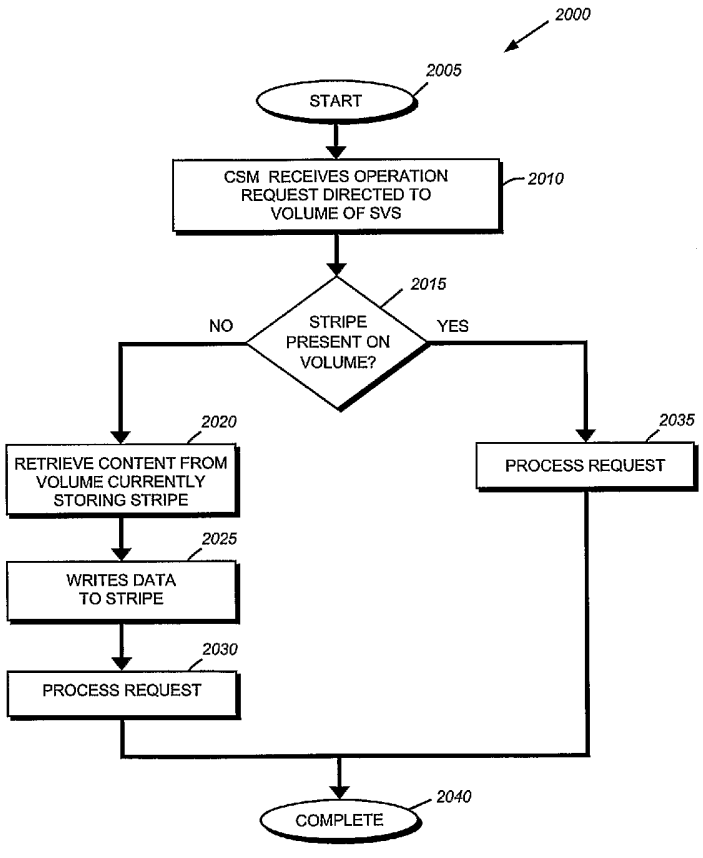
(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, 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, 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: SYSTEM AND METHOD FOR RESTRIPING DATA ACROSS A PLURALITY OF VOLUMES




(57) Abstract: A system and method re-stripes one or more data containers across a striped volume set (SVS) that has been modified by the addition of one or more volumes. The SVS is associated with an existing set of striping rules that define a stripe algorithm, a stripe width and an ordered list of volumes distributed across a plurality of nodes interconnected as a cluster. Each node of the cluster includes (i) a disk element (D-blade) adapted to service a volume of the SVS and (ii) a network element (N-blade) adapted to redirect a data access request to any D-blade of the cluster. Notably, the content of each data container is apportioned among the volumes of the SVS to thereby improve the effi- ciency of storage service provided by the cluster. To that end, the stripe algorithm specifies the manner in which the data container content is apportioned as stripes across the plurality of volumes, while the stripe width specifies the size/width of each stripe.

WO 2006/118926 A2

# SYSTEM AND METHOD FOR RESTRIPING DATA ACROSS A PLURALITY OF VOLUMES

## FIELD OF THE INVENTION

The present invention relates to file systems and, more particularly, to restriping a data across a plurality of volumes in a file system.

## BACKGROUND OF THE INVENTION

A storage system typically comprises one or more storage devices into which information may be entered, and from which information may be obtained, as desired. The storage system includes a storage operating system that functionally organizes the system by, *inter alia*, invoking storage operations in support of a storage service implemented by the system. The storage system may be implemented in accordance with a variety of storage architectures including, but not limited to, a network-attached storage environment, a storage area network and a disk assembly directly attached to a client or host computer. The storage devices are typically disk drives organized as a disk array, wherein the term "disk" commonly describes a self-contained rotating magnetic media storage device. The term disk in this context is synonymous with hard disk drive (HDD) or direct access storage device (DASD).

The storage operating system of the storage system may implement a high-level module, such as a file system, to logically organize the information stored on volumes as a hierarchical structure of data containers, such as files and logical units. For example, each "on-disk" file may be implemented as set of data structures, i.e., disk blocks, configured to store information, such as the actual data for the file. These data blocks are organized within a volume block number (vbn) space that is maintained by the file system. The file system may also assign each data block in the file a corresponding "file offset" or file block number (fbn). The file system typically assigns sequences of fbns on a per-file basis, whereas vbns are assigned over a larger volume address space. The file

system organizes the data blocks within the vbn space as a "logical volume"; each logical volume may be, although is not necessarily, associated with its own file system.

A known type of file system is a write-anywhere file system that does not overwrite data on disks. If a data block is retrieved (read) from disk into a memory of the storage system and "dirtied" (i.e., updated or modified) with new data, the data block is thereafter stored (written) to a new location on disk to optimize write performance. A write-anywhere file system may initially assume an optimal layout such that the data is substantially contiguously arranged on disks. The optimal disk layout results in efficient access operations, particularly for sequential read operations, directed to the disks. An example of a write-anywhere file system that is configured to operate on a storage system is the Write Anywhere File Layout (WAFL™) file system available from Network Appliance, Inc., Sunnyvale, California.

The storage system may be further configured to operate according to a client/server model of information delivery to thereby allow many clients to access data containers stored on the system. In this model, the client may comprise an application, such as a database application, executing on a computer that "connects" to the storage system over a computer network, such as a point-to-point link, shared local area network (LAN), wide area network (WAN), or virtual private network (VPN) implemented over a public network such as the Internet. Each client may request the services of the storage system by issuing file-based and block-based protocol messages (in the form of packets) to the system over the network.

A plurality of storage systems may be interconnected to provide a storage system environment configured to service many clients. Each storage system may be configured to service one or more volumes, wherein each volume stores one or more data containers. Yet often a large number of data access requests issued by the clients may be directed to a small number of data containers serviced by a particular storage system of the environment. A solution to such a problem is to distribute the volumes serviced by the particular storage system among all of the storage systems of the environment. This, in turn, distributes the data access requests, along with the processing resources needed to service such requests, among all of the storage systems, thereby reducing the individual process-

ing load on each storage system. However, a noted disadvantage arises when only a single data container, such as a file, is heavily accessed by clients of the storage system environment. As a result, the storage system attempting to service the requests directed to that data container may exceed its processing resources and become overburdened, with a concomitant degradation of speed and performance.

One technique for overcoming the disadvantages of having a single data container that is heavily utilized is to stripe the data container across a plurality of volumes configured as a striped volume set, where each volume is serviced by a different storage system, thereby distributing the load for the single data container among a plurality of storage systems. One technique for data container striping is described in the above-referenced U.S. Patent Application Serial No. 11/119,278, entitled STORAGE SYSTEM ARCHITECTURE FOR STRIPING DATA CONTAINER CONTENT ACROSS VOLUMES OF A CLUSTER. However, a noted disadvantage of such a file striping technique arises when the number of volumes within a striped volume set changes. For example, a striped volume set may initially be generated with three volumes populated with data; subsequently, a fourth volume may be added to the striped volume set. Moreover, additional volumes may be added to the striped volume set to enable distribution of the load among an even greater number of storage systems.

A conventional "brute force" re-striping technique operates to first copy all of the data from an existing striped volume set to a temporary holding storage data container. A new striped volume set is configured that includes the additional volumes and all of the data is then copied to the newly configured striped volume set. As can be appreciated, the computational overhead required for such a brute force re-striping operation is substantial. Additionally, there may be times during the re-striping process when data is not available for clients. Another noted disadvantage of such a re-striping technique is that it requires a second data storage container of sufficient size to hold the entire contents of the existing striped volume set. As this second data storage container is only utilized during the brief period of the striped volume set reconfiguration, the conventional brute force re-striping technique is extremely costly in terms of storage space utilization.

## SUMMARY OF THE INVENTION

The present invention overcomes the disadvantages of the prior art by providing a system and method for re-striping one or more data containers across a striped volume set (SVS) that has been modified by the addition or removal of one or more volumes. The SVS is associated with an existing set of striping rules that define, *inter alia*, a stripe algorithm, a stripe width and an ordered list of volumes distributed across a plurality of nodes interconnected as a cluster. Each node of the cluster includes (i) a disk element (D-blade) adapted to service a volume of the SVS and (ii) a network element (N-blade) adapted to redirect a data access request to any D-blade of the cluster. Notably, the content of each data container is apportioned among the volumes of the SVS to thereby improve the efficiency of storage service provided by the cluster. To that end, the stripe algorithm specifies the manner in which the data container content is apportioned as stripes across the plurality of volumes, while the stripe width specifies the size/width of each stripe.

According to a first embodiment of the invention, a new set of striping rules is created in response to a re-striping operation request received by an N-blade from, e.g., an administrator desiring to increase the number of volumes in the SVS. Although the stripe algorithm and stripe width may remain the same, the new set of striping rules incorporates the additional volumes as constituents of the ordered list of volumes in the SVS. The new set of striping rules is then marked as "new" and installed in a SVS entry of a volume location database (VLDB). Any subsequent data access request received at an N-blade of a node is processed in accordance with the new striping rules. In addition, the existing set of striping rules is marked "old".

The N-blade then initiates the re-striping operation by instructing each D-blade of each node about the re-striping request. Each D-blade includes a container striping module (VSM) configured to implement a Locate() function that computes the location of data container content (i.e., a stripe) in its SVS volume. Accordingly, the VSM of each D-blade determines whether the volume holds a stripe of the data container affected by the re-striping operation and, if so, re-locates the content of the stripe to the additional volume in accordance with the new striping rules. Data relocation illustratively occurs as

a background procedure to ensure that D-blades prioritize servicing of subsequent data access requests issued by clients. If a subsequent data access request is forwarded to a D-blade of an additional volume that has yet to receive the re-located content of a stripe, the VSM of the D-blade serving that volume uses the old set of striping rules to retrieve the data from the volume currently storing the stripe so that it may promptly service the request. Upon completion of data relocation among the additional volumes, the old set of striping rules is deleted from the SVS entry of the VLDB.

In a second embodiment of the invention, the SVS is associated with multiple (e.g., new and old) sets of striping rules, each defining a stripe algorithm, a stripe width and an ordered list of volumes distributed across a plurality of nodes in the cluster. Here, the new set of striping rules is used for any newly created data containers stored on the SVS, while the old set of striping rules is used for all existing data containers stored on the SVS. Essentially, each data container in the SVS is associated with a set of striping rules. This second embodiment of the invention advantageously enables substantially instantaneous re-striping of new data containers without disrupting the existing striped containers.

According to an aspect of the invention, the association between a data container and set of striping rules is illustratively effected through the use of an additional field added to a data container handle used to access the data container in the node. The data container handle is convenient for rendering such an association because it is accessible by the N-blade, which needs to know the appropriate set of striping rules for a data container when determining to which D-blade (and volume) to re-direct a data access request. Alternatively, a generation value of the data container handle can be manipulated (i.e., "overloaded") to identify the set of striping rules to be applied to the data container.

## BRIEF DESCRIPTION OF THE DRAWINGS

The above and further advantages of invention may be better understood by referring to the following description in conjunction with the accompanying drawings in which like reference numerals indicate identical or functionally similar elements:

Fig. 1 is a schematic block diagram of a plurality of nodes interconnected as a cluster in accordance with an embodiment of the present invention;

Fig. 2 is a schematic block diagram of a node in accordance with an embodiment of the present invention;

Fig. 3 is a schematic block diagram of a storage operating system that may be advantageously used with the present invention;

Fig. 4 is a schematic block diagram illustrating the format of a cluster fabric (CF) message in accordance with an embodiment of with the present invention;

Fig. 5 is a schematic block diagram illustrating the format of a data container handle in accordance with an embodiment of the present invention;

Fig. 6 is a schematic block diagram of an exemplary inode in accordance with an embodiment of the present invention;

Fig. 7 is a schematic block diagram of an exemplary buffer tree in accordance with an embodiment of the present invention;

Fig. 8 is a schematic block diagram of an illustrative embodiment of a buffer tree of a file that may be advantageously used with the present invention;

Fig. 9 is a schematic block diagram of an exemplary aggregate in accordance with an embodiment of the present invention;

Fig. 10 is a schematic block diagram of an exemplary on-disk layout of the aggregate in accordance with an embodiment of the present invention;

Fig. 11 is a schematic block diagram illustrating a collection of management processes in accordance with an embodiment of the present invention;

Fig. 12 is a schematic block diagram of a volume location database (VLDB) volume entry in accordance with an embodiment of the present invention;

Fig. 13 is a schematic block diagram of a VLDB aggregate entry in accordance with an embodiment of the present invention;

Fig. 14 is a schematic block diagram of a striped volume set (SVS) in accordance with an embodiment of the present invention;

Fig. 15 is a schematic block diagram of a VLDB SVS entry in accordance with an embodiment the present invention;

Fig. 16 is a schematic block diagram illustrating the periodic sparseness of file content stored on volumes of a SVS in accordance with an embodiment of the present invention;

Fig. 17 is a flow chart detailing the steps of a procedure for performing a re-striping operation in accordance with a first embodiment of the present invention;

Fig. 18 is a schematic block diagram of an exemplary SVS after the addition of a volume to the SVS of Fig. 16 in accordance with an embodiment of the present invention;

Fig. 19 is a schematic block diagram of an exemplary SVS after completion of the re-striping procedure of Fig. 17 in accordance with an embodiment of the present invention;

Fig. 20 is a flowchart detailing the steps of a procedure for processing a data access request directed to a volume of a SVS prior to completion of a re-striping operation in accordance with an embodiment of the present invention;

Fig. 21 is a flow chart detailing the steps of a background procedure for relocating data in accordance with an embodiment of the present invention;

Fig. 22 is a schematic block diagram of the format of a data container handle that is modified to include a striping rule set identifier field in accordance with an embodiment of the present invention;

Fig. 23 is a flow chart detailing the steps of a procedure for performing a re-striping operation in accordance with a second embodiment of the present invention; and

Fig. 24 is a schematic block diagram of an exemplary SVS after the re-striping operation in accordance with the second embodiment of the present invention.

# DETAILED DESCRIPTION OF AN ILLUSTRATIVE EMBODIMENT

A. Cluster Environment

Fig. 1 is a schematic block diagram of a plurality of nodes 200 interconnected as a cluster 100 and configured to provide storage service relating to the organization of information on storage devices. The nodes 200 comprise various functional components that cooperate to provide a distributed storage system architecture of the cluster 100. To that end, each node 200 is generally organized as a network element (N-blade 310) and a disk element (D-blade 350). The N-blade 310 includes functionality that enables the node 200 to connect to clients 180 over a computer network 140, while each D-blade 350 connects to one or more storage devices, such as disks 130 of a disk array 120. The nodes 200 are interconnected by a cluster switching fabric 150 which, in the illustrative embodiment, may be embodied as a Gigabit Ethernet switch. An exemplary distributed file system architecture is generally described in U.S. Patent Application Publication No. US 2002/0116593 titled METHOD AND SYSTEM FOR RESPONDING TO FILE SYSTEM REQUESTS, by M. Kazar et al. published August 22, 2002. It should be noted that while there is shown an equal number of N and D-blades in the illustrative cluster 100, there may be differing numbers of N and/or D-blades in accordance with various embodiments of the present invention. For example, there may be a plurality of N-blades and/or D-blades interconnected in a cluster configuration 100 that does not reflect a one-to-one correspondence between the N and D-blades. As such, the description of a node 200 comprising one N-blade and one D-blade should be taken as illustrative only.

The clients 180 may be general-purpose computers configured to interact with the node 200 in accordance with a client/server model of information delivery. That is, each client may request the services of the node, and the node may return the results of the services requested by the client, by exchanging packets over the network 140. The client may issue packets including file-based access protocols, such as the Common Internet File System (CIFS) protocol or Network File System (NFS) protocol, over the Transmission Control Protocol/Internet Protocol (TCP/IP) when accessing information in the form of files and directories. Alternatively, the client may issue packets including block-based

access protocols, such as the Small Computer Systems Interface (SCSI) protocol encapsulated over TCP (iSCSI) and SCSI encapsulated over Fibre Channel (FCP), when accessing information in the form of blocks.

B. Storage System Node

Fig. 2 is a schematic block diagram of a node 200 that is illustratively embodied as a storage system comprising a plurality of processors 222a,b, a memory 224, a network adapter 225, a cluster access adapter 226, a storage adapter 228 and local storage 230 interconnected by a system bus 223. The local storage 230 comprises one or more storage devices, such as disks, utilized by the node to locally store configuration information (e.g., in configuration table 235) provided by one or more management processes that execute as user mode applications 1100 (see Fig. 11). The cluster access adapter 226 comprises a plurality of ports adapted to couple the node 200 to other nodes of the cluster 100. In the illustrative embodiment, Ethernet is used as the clustering protocol and interconnect media, although it will be apparent to those skilled in the art that other types of protocols and interconnects may be utilized within the cluster architecture described herein. In alternate embodiments where the N-blades and D-blades are implemented on separate storage systems or computers, the cluster access adapter 226 is utilized by the N/D-blade for communicating with other N/D-blades in the cluster 100.

Each node 200 is illustratively embodied as a dual processor storage system executing a storage operating system 300 that preferably implements a high-level module, such as a file system, to logically organize the information as a hierarchical structure of named directories, files and special types of files called virtual disks (hereinafter generally "blocks") on the disks. However, it will be apparent to those of ordinary skill in the art that the node 200 may alternatively comprise a single or more than two processor system. Illustratively, one processor 222a executes the functions of the N-blade 310 on the node, while the other processor 222b executes the functions of the D-blade 350.

The memory 224 illustratively comprises storage locations that are addressable by the processors and adapters for storing software program code and data structures associated with the present invention. The processor and adapters may, in turn, comprise processing elements and/or logic circuitry configured to execute the software code and ma-

nipulate the data structures. The storage operating system 300, portions of which is typically resident in memory and executed by the processing elements, functionally organizes the node 200 by, *inter alia*, invoking storage operations in support of the storage service implemented by the node. It will be apparent to those skilled in the art that other processing and memory means, including various computer readable media, may be used for storing and executing program instructions pertaining to the invention described herein.

The network adapter 225 comprises a plurality of ports adapted to couple the node 200 to one or more clients 180 over point-to-point links, wide area networks, virtual private networks implemented over a public network (Internet) or a shared local area network. The network adapter 225 thus may comprise the mechanical, electrical and signaling circuitry needed to connect the node to the network. Illustratively, the computer network 140 may be embodied as an Ethernet network or a Fibre Channel (FC) network. Each client 180 may communicate with the node over network 140 by exchanging discrete frames or packets of data according to pre-defined protocols, such as TCP/IP.

The storage adapter 228 cooperates with the storage operating system 300 executing on the node 200 to access information requested by the clients. The information may be stored on any type of attached array of writable storage device media such as video tape, optical, DVD, magnetic tape, bubble memory, electronic random access memory, micro-electro mechanical and any other similar media adapted to store information, including data and parity information. However, as illustratively described herein, the information is preferably stored on the disks 130 of array 120. The storage adapter comprises a plurality of ports having input/output (I/O) interface circuitry that couples to the disks over an I/O interconnect arrangement, such as a conventional high-performance, FC link topology.

Storage of information on each array 120 is preferably implemented as one or more storage "volumes" that comprise a collection of physical storage disks 130 cooperating to define an overall logical arrangement of volume block number (vbn) space on the volume(s). Each logical volume is generally, although not necessarily, associated with its own file system. The disks within a logical volume/file system are typically organized as one or more groups, wherein each group may be operated as a Redundant Array of Inde-

pendent (or Inexpensive) Disks (RAID). Most RAID implementations, such as a RAID-4 level implementation, enhance the reliability/integrity of data storage through the redundant writing of data "stripes" across a given number of physical disks in the RAID group, and the appropriate storing of parity information with respect to the striped data. An illustrative example of a RAID implementation is a RAID-4 level implementation, although it should be understood that other types and levels of RAID implementations may be used in accordance with the inventive principles described herein.

C. Storage Operating System

To facilitate access to the disks 130, the storage operating system 300 implements a write-anywhere file system that cooperates with one or more virtualization modules to "virtualize" the storage space provided by disks 130. The file system logically organizes the information as a hierarchical structure of named directories and files on the disks. Each "on-disk" file may be implemented as set of disk blocks configured to store information, such as data, whereas the directory may be implemented as a specially formatted file in which names and links to other files and directories are stored. The virtualization module(s) allow the file system to further logically organize information as a hierarchical structure of blocks on the disks that are exported as named logical unit numbers (luns).

In the illustrative embodiment, the storage operating system is preferably the NetApp® Data ONTAP™ operating system available from Network Appliance, Inc., Sunnyvale, California that implements a Write Anywhere File Layout (WAFL™) file system. However, it is expressly contemplated that any appropriate storage operating system may be enhanced for use in accordance with the inventive principles described herein. As such, where the term "WAFL" is employed, it should be taken broadly to refer to any storage operating system that is otherwise adaptable to the teachings of this invention.

Fig. 3 is a schematic block diagram of the storage operating system 300 that may be advantageously used with the present invention. The storage operating system comprises a series of software layers organized to form an integrated network protocol stack or, more generally, a multi-protocol engine 325 that provides data paths for clients to access information stored on the node using block and file access protocols. The multi-protocol engine includes a media access layer 312 of network drivers (e.g., gigabit

Ethernet drivers) that interfaces to network protocol layers, such as the IP layer 314 and its supporting transport mechanisms, the TCP layer 316 and the User Datagram Protocol (UDP) layer 315. A file system protocol layer provides multi-protocol file access and, to that end, includes support for the Direct Access File System (DAFS) protocol 318, the NFS protocol 320, the CIFS protocol 322 and the Hypertext Transfer Protocol (HTTP) protocol 324. A VI layer 326 implements the VI architecture to provide direct access transport (DAT) capabilities, such as RDMA, as required by the DAFS protocol 318. An iSCSI driver layer 328 provides block protocol access over the TCP/IP network protocol layers, while a FC driver layer 330 receives and transmits block access requests and responses to and from the node. The FC and iSCSI drivers provide FC-specific and iSCSI-specific access control to the blocks and, thus, manage exports of luns to either iSCSI or FCP or, alternatively, to both iSCSI and FCP when accessing the blocks on the node 200.

In addition, the storage operating system includes a series of software layers organized to form a storage server 365 that provides data paths for accessing information stored on the disks 130 of the node 200. To that end, the storage server 365 includes a file system module 360 in cooperating relation with a volume striping module (VSM) 370, a RAID system module 380 and a disk driver system module 390. The RAID system 380 manages the storage and retrieval of information to and from the volumes/disks in accordance with I/O operations, while the disk driver system 390 implements a disk access protocol such as, e.g., the SCSI protocol. The VSM 370 illustratively implements a striped volume set (SVS) of the present invention. As described further herein, the VSM cooperates with the file system 360 to enable storage server 365 to service a volume of the SVS. In particular, the VSM 370 implements the novel Locate() function 375 to compute the location of data container content in the SVS volume to thereby ensure consistency of such content served by the cluster.

The file system 360 implements a virtualization system of the storage operating system 300 through the interaction with one or more virtualization modules illustratively embodied as, e.g., a virtual disk (vdisk) module (not shown) and a SCSI target module 335. The vdisk module enables access by administrative interfaces, such as a user interface of a management framework 1110 (see Fig. 11), in response to a user (system administrator) issuing commands to the node 200. The SCSI target module 335 is generally

disposed between the FC and iSCSI drivers 328, 330 and the file system 360 to provide a translation layer of the virtualization system between the block (lun) space and the file system space, where luns are represented as blocks.

The file system 360 is illustratively a message-based system that provides logical volume management capabilities for use in access to the information stored on the storage devices, such as disks. That is, in addition to providing file system semantics, the file system 360 provides functions normally associated with a volume manager. These functions include (i) aggregation of the disks, (ii) aggregation of storage bandwidth of the disks, and (iii) reliability guarantees, such as mirroring and/or parity (RAID). The file system 360 illustratively implements the WAFL file system (hereinafter generally the "write-anywhere file system") having an on-disk format representation that is block-based using, e.g., 4 kilobyte (kB) blocks and using index nodes ("inodes") to identify files and file attributes (such as creation time, access permissions, size and block location). The file system uses files to store meta-data describing the layout of its file system; these meta-data files include, among others, an inode file. A file handle, i.e., an identifier that includes an inode number, is used to retrieve an inode from disk.

Broadly stated, all inodes of the write-anywhere file system are organized into the inode file. A file system (fs) info block specifies the layout of information in the file system and includes an inode of a file that includes all other inodes of the file system. Each logical volume (file system) has an fsinfo block that is preferably stored at a fixed location within, e.g., a RAID group. The inode of the inode file may directly reference (point to) data blocks of the inode file or may reference indirect blocks of the inode file that, in turn, reference data blocks of the inode file. Within each data block of the inode file are embedded inodes, each of which may reference indirect blocks that, in turn, reference data blocks of a file.

Operationally, a request from the client 180 is forwarded as a packet over the computer network 140 and onto the node 200 where it is received at the network adapter 225. A network driver (of layer 312 or layer 330) processes the packet and, if appropriate, passes it on to a network protocol and file access layer for additional processing prior to forwarding to the write-anywhere file system 360. Here, the file system generates op-

erations to load (retrieve) the requested data from disk 130 if it is not resident "in core", i.e., in memory 224. If the information is not in memory, the file system 360 indexes into the inode file using the inode number to access an appropriate entry and retrieve a logical vbn. The file system then passes a message structure including the logical vbn to the RAID system 380; the logical vbn is mapped to a disk identifier and disk block number (disk,dbn) and sent to an appropriate driver (e.g., SCSI) of the disk driver system 390. The disk driver accesses the dbn from the specified disk 130 and loads the requested data block(s) in memory for processing by the node. Upon completion of the request, the node (and operating system) returns a reply to the client 180 over the network 140.

It should be noted that the software "path" through the storage operating system layers described above needed to perform data storage access for the client request received at the node may alternatively be implemented in hardware. That is, in an alternate embodiment of the invention, a storage access request data path may be implemented as logic circuitry embodied within a field programmable gate array (FPGA) or an application specific integrated circuit (ASIC). This type of hardware implementation increases the performance of the storage service provided by node 200 in response to a request issued by client 180. Moreover, in another alternate embodiment of the invention, the processing elements of adapters 225, 228 may be configured to offload some or all of the packet processing and storage access operations, respectively, from processor 222, to thereby increase the performance of the storage service provided by the node. It is expressly contemplated that the various processes, architectures and procedures described herein can be implemented in hardware, firmware or software.

As used herein, the term "storage operating system" generally refers to the computer-executable code operable on a computer to perform a storage function that manages data access and may, in the case of a node 200, implement data access semantics of a general purpose operating system. The storage operating system can also be implemented as a microkernel, an application program operating over a general-purpose operating system, such as UNIX® or Windows NT®, or as a general-purpose operating system with configurable functionality, which is configured for storage applications as described herein.

In addition, it will be understood to those skilled in the art that the invention described herein may apply to any type of special-purpose (e.g., file server, filer or storage serving appliance) or general-purpose computer, including a standalone computer or portion thereof, embodied as or including a storage system. Moreover, the teachings of this invention can be adapted to a variety of storage system architectures including, but not limited to, a network-attached storage environment, a storage area network and disk assembly directly-attached to a client or host computer. The term "storage system" should therefore be taken broadly to include such arrangements in addition to any subsystems configured to perform a storage function and associated with other equipment or systems. It should be noted that while this description is written in terms of a write any where file system, the teachings of the present invention may be utilized with any suitable file system, including a write in place file system.

D. CF Protocol

In the illustrative embodiment, the storage server 365 is embodied as D-blade 350 of the storage operating system 300 to service one or more volumes of array 120. In addition, the multi-protocol engine 325 is embodied as N-blade 310 to (i) perform protocol termination with respect to a client issuing incoming data access request packets over the network 140, as well as (ii) redirect those data access requests to any storage server 365 of the cluster 100. Moreover, the N-blade 310 and D-blade 350 cooperate to provide a highly-scalable, distributed storage system architecture of the cluster 100. To that end, each blade includes a cluster fabric (CF) interface module 340a,b adapted to implement intra-cluster communication among the blades, including D-blade-to-D-blade communication for data container striping operations described herein.

The protocol layers, e.g., the NFS/CIFS layers and the iSCSI/FC layers, of the N-blade 310 function as protocol servers that translate file-based and block based data access requests from clients into CF protocol messages used for communication with the D-blade 350. That is, the N-blade servers convert the incoming data access requests into file system primitive operations (commands) that are embedded within CF messages by the CF interface module 340 for transmission to the D-blades 350 of the cluster 100. Notably, the CF interface modules 340 cooperate to provide a single file system image

across all D-blades 350 in the cluster 100. Thus, any network port of an N-blade that receives a client request can access any data container within the single file system image located on any D-blade 350 of the cluster.

Further to the illustrative embodiment, the N-blade 310 and D-blade 350 are implemented as separately-scheduled processes of storage operating system 300; however, in an alternate embodiment, the blades may be implemented as pieces of code within a single operating system process. Communication between an N-blade and D-blade is thus illustratively effected through the use of message passing between the blades although, in the case of remote communication between an N-blade and D-blade of different nodes, such message passing occurs over the cluster switching fabric 150. A known message-passing mechanism provided by the storage operating system to transfer information between blades (processes) is the Inter Process Communication (IPC) mechanism. The protocol used with the IPC mechanism is illustratively a generic file and/or block-based "agnostic" CF protocol that comprises a collection of methods/functions constituting a CF application programming interface (API). Examples of such an agnostic protocol are the SpinFS and SpinNP protocols available from Network Appliance, Inc. The SpinFS protocol is described in the above-referenced U.S. Patent Application Publication No. US 2002/0116593.

The CF interface module 340 implements the CF protocol for communicating file system commands among the blades of cluster 100. Communication is illustratively effected by the D-blade exposing the CF API to which an N-blade (or another D-blade) issues calls. To that end, the CF interface module 340 is organized as a CF encoder and CF decoder. The CF encoder of, e.g., CF interface 340a on N-blade 310 encapsulates a CF message as (i) a local procedure call (LPC) when communicating a file system command to a D-blade 350 residing on the same node 200 or (ii) a remote procedure call (RPC) when communicating the command to a D-blade residing on a remote node of the cluster 100. In either case, the CF decoder of CF interface 340b on D-blade 350 de-encapsulates the CF message and processes the file system command.

Fig. 4 is a schematic block diagram illustrating the format of a CF message 400 in accordance with an embodiment of with the present invention. The CF message 400 is

illustratively used for RPC communication over the switching fabric 150 between remote blades of the cluster 100; however, it should be understood that the term "CF message" may be used generally to refer to LPC and RPC communication between blades of the cluster. The CF message 400 includes a media access layer 402, an IP layer 404, a UDP layer 406, a reliable connection (RC) layer 408 and a CF protocol layer 410. As noted, the CF protocol is a generic file system protocol that conveys file system commands related to operations contained within client requests to access data containers stored on the cluster 100; the CF protocol layer 410 is that portion of message 400 that carries the file system commands. Illustratively, the CF protocol is datagram based and, as such, involves transmission of messages or "envelopes" in a reliable manner from a source (e.g., an N-blade 310) to a destination (e.g., a D-blade 350). The RC layer 408 implements a reliable transport protocol that is adapted to process such envelopes in accordance with a connectionless protocol, such as UDP 406.

A data container, e.g., a file, is accessed in the file system using a data container handle. Fig. 5 is a schematic block diagram illustrating the format of a data container handle 500 including a SVS ID field 502, an inode number field 504, a unique-ifier field 506 a striped flag field 508 and a striping epoch number field 510. The SVS ID field 502 contains a global identifier (within the cluster 100) of the SVS within which the data container resides. The inode number field 504 contains an inode number of an inode (within an inode file) pertaining to the data container. The unique-ifier field 506 contains a monotonically increasing number that uniquely identifies the data container handle 500. The unique-ifier is particularly useful in the case where an inode number has been deleted, reused and reassigned to a new data container. The unique-ifier distinguishes that reused inode number in a particular data container from a potentially previous use of those fields. The striped flag field 508 is illustratively a Boolean value that identifies whether the data container is striped or not. The striping epoch number field 510 indicates the appropriate striping technique for use with this data container for embodiments where the SVS utilizes differing striping techniques for different data containers.

E. File System Organization

In the illustrative embodiment, a data container is represented in the write-anywhere file system as an inode data structure adapted for storage on the disks 130. Fig. 6 is a schematic block diagram of an inode 600, which preferably includes a meta-data section 605 and a data section 660. The information stored in the meta-data section 605 of each inode 600 describes the data container (e.g., a file) and, as such, includes the type (e.g., regular, directory, vdisk) 610 of file, its size 615, time stamps (e.g., access and/or modification time) 620 and ownership, i.e., user identifier (UID 625) and group ID (GID 630), of the file. The meta-data section 605 also includes a generation number 631, and a meta-data invalidation flag field 634. As described further herein, meta-data invalidation flag field 634 is used to indicate whether meta-data in this inode is usable or whether it should be re-acquired from the MDV. The contents of the data section 660 of each inode may be interpreted differently depending upon the type of file (inode) defined within the type field 610. For example, the data section 660 of a directory inode contains meta-data controlled by the file system, whereas the data section of a regular inode contains file system data. In this latter case, the data section 660 includes a representation of the data associated with the file.

Specifically, the data section 660 of a regular on-disk inode may include file system data or pointers, the latter referencing 4kB data blocks on disk used to store the file system data. Each pointer is preferably a logical vbn to facilitate efficiency among the file system and the RAID system 380 when accessing the data on disks. Given the restricted size (e.g., 128 bytes) of the inode, file system data having a size that is less than or equal to 64 bytes is represented, in its entirety, within the data section of that inode. However, if the length of the contents of the data container exceeds 64 bytes but less than or equal to 64kB, then the data section of the inode (e.g., a first level inode) comprises up to 16 pointers, each of which references a 4kB block of data on the disk.

Moreover, if the size of the data is greater than 64kB but less than or equal to 64 megabytes (MB), then each pointer in the data section 660 of the inode (e.g., a second level inode) references an indirect block (e.g., a first level L1 block) that contains 1024 pointers, each of which references a 4kB data block on disk. For file system data having a size greater than 64MB, each pointer in the data section 660 of the inode (e.g., a third level L3 inode) references a double-indirect block (e.g., a second level L2 block) that

contains 1024 pointers, each referencing an indirect (e.g., a first level L1) block. The indirect block, in turn, that contains 1024 pointers, each of which references a 4kB data block on disk. When accessing a file, each block of the file may be loaded from disk 130 into the memory 224.

When an on-disk inode (or block) is loaded from disk 130 into memory 224, its corresponding in-core structure embeds the on-disk structure. For example, the dotted line surrounding the inode 600 indicates the in-core representation of the on-disk inode structure. The in-core structure is a block of memory that stores the on-disk structure plus additional information needed to manage data in the memory (but not on disk). The additional information may include, e.g., a "dirty" bit 670. After data in the inode (or block) is updated/modified as instructed by, e.g., a write operation, the modified data is marked "dirty" using the dirty bit 670 so that the inode (block) can be subsequently "flushed" (stored) to disk. The in-core and on-disk format structures of the WAFL file system, including the inodes and inode file, are disclosed and described in the previously incorporated U.S. Patent No. 5,819,292 titled METHOD FOR MAINTAINING CONSISTENT STATES OF A FILE SYSTEM AND FOR CREATING USER-ACCESSIBLE READ-ONLY COPIES OF A FILE SYSTEM by David Hitz et al., issued on October 6, 1998.

Fig. 7 is a schematic block diagram of an embodiment of a buffer tree of a file that may be advantageously used with the present invention. The buffer tree is an internal representation of blocks for a file (e.g., file 700) loaded into the memory 224 and maintained by the write-anywhere file system 360. A root (top-level) inode 702, such as an embedded inode, references indirect (e.g., level 1) blocks 704. Note that there may be additional levels of indirect blocks (e.g., level 2, level 3) depending upon the size of the file. The indirect blocks (and inode) contain pointers 705 that ultimately reference data blocks 706 used to store the actual data of the file. That is, the data of file 700 are contained in data blocks and the locations of these blocks are stored in the indirect blocks of the file. Each level 1 indirect block 704 may contain pointers to as many as 1024 data blocks. According to the "write anywhere" nature of the file system, these blocks may be located anywhere on the disks 130.

A file system layout is provided that apportions an underlying physical volume into one or more virtual volumes (or flexible volume) of a storage system, such as node 200. An example of such a file system layout is described in U.S. Patent Application Serial No. 10/836,817 titled EXTENSION OF WRITE ANYWHERE FILE SYSTEM LAYOUT, by John K. Edwards et al. and assigned to Network Appliance, Inc. The underlying physical volume is an aggregate comprising one or more groups of disks, such as RAID groups, of the node. The aggregate has its own physical volume block number (pvbn) space and maintains meta-data, such as block allocation structures, within that pvbn space. Each flexible volume has its own virtual volume block number (vvbn) space and maintains meta-data, such as block allocation structures, within that vvbn space. Each flexible volume is a file system that is associated with a container file; the container file is a file in the aggregate that contains all blocks used by the flexible volume. Moreover, each flexible volume comprises data blocks and indirect blocks that contain block pointers that point at either other indirect blocks or data blocks.

In one embodiment, pvbns are used as block pointers within buffer trees of files (such as file 700) stored in a flexible volume. This "hybrid" flexible volume embodiment involves the insertion of only the pvbn in the parent indirect block (e.g., inode or indirect block). On a read path of a logical volume, a "logical" volume (vol) info block has one or more pointers that reference one or more fsinfo blocks, each of which, in turn, points to an inode file and its corresponding inode buffer tree. The read path on a flexible volume is generally the same, following pvbns (instead of vvbns) to find appropriate locations of blocks; in this context, the read path (and corresponding read performance) of a flexible volume is substantially similar to that of a physical volume. Translation from pvbn-to-disk,dbn occurs at the file system/RAID system boundary of the storage operating system 300.

In an illustrative dual vbn hybrid flexible volume embodiment, both a pvbn and its corresponding vvbn are inserted in the parent indirect blocks in the buffer tree of a file. That is, the pvbn and vvbn are stored as a pair for each block pointer in most buffer tree structures that have pointers to other blocks, e.g., level 1(L1) indirect blocks, inode file level 0 (L0) blocks. Fig. 8 is a schematic block diagram of an illustrative embodiment of a buffer tree of a file 800 that may be advantageously used with the present in-

vention. A root (top-level) inode 802, such as an embedded inode, references indirect (e.g., level 1) blocks 804. Note that there may be additional levels of indirect blocks (e.g., level 2, level 3) depending upon the size of the file. The indirect blocks (and inode) contain pvbn/vvbn pointer pair structures 808 that ultimately reference data blocks 806 used to store the actual data of the file.

The pvbns reference locations on disks of the aggregate, whereas the vvbns reference locations within files of the flexible volume. The use of pvbns as block pointers 808 in the indirect blocks 804 provides efficiencies in the read paths, while the use of vvbn block pointers provides efficient access to required meta-data. That is, when freeing a block of a file, the parent indirect block in the file contains readily available vvbn block pointers, which avoids the latency associated with accessing an owner map to perform pvbn-to-vvbn translations; yet, on the read path, the pvbn is available.

Fig. 9 is a schematic block diagram of an embodiment of an aggregate 900 that may be advantageously used with the present invention. Luns (blocks) 902, directories 904, qtrees 906 and files 908 may be contained within flexible volumes 910, such as dual vbn flexible volumes, that, in turn, are contained within the aggregate 900. The aggregate 900 is illustratively layered on top of the RAID system, which is represented by at least one RAID plex 950 (depending upon whether the storage configuration is mirrored), wherein each plex 950 comprises at least one RAID group 960. Each RAID group further comprises a plurality of disks 930, e.g., one or more data (D) disks and at least one (P) parity disk.

Whereas the aggregate 900 is analogous to a physical volume of a conventional storage system, a flexible volume is analogous to a file within that physical volume. That is, the aggregate 900 may include one or more files, wherein each file contains a flexible volume 910 and wherein the sum of the storage space consumed by the flexible volumes is physically smaller than (or equal to) the size of the overall physical volume. The aggregate utilizes a *physical* pvbn space that defines a storage space of blocks provided by the disks of the physical volume, while each embedded flexible volume (within a file) utilizes a *logical* vvbn space to organize those blocks, e.g., as files. Each vvbn space is an independent set of numbers that corresponds to locations within the file, which loca-

tions are then translated to dbns on disks. Since the flexible volume 910 is also a logical volume, it has its own block allocation structures (e.g., active, space and summary maps) in its vvbn space.

A container file is a file in the aggregate that contains all blocks used by a flexible volume. The container file is an internal (to the aggregate) feature that supports a flexible volume; illustratively, there is one container file per flexible volume. Similar to a pure logical volume in a file approach, the container file is a hidden file (not accessible to a user) in the aggregate that holds every block in use by the flexible volume. The aggregate includes an illustrative hidden meta-data root directory that contains subdirectories of flexible volumes:

*WAFL/fsid/filesystem file, storage label file*

Specifically, a physical file system (WAFL) directory includes a subdirectory for each flexible volume in the aggregate, with the name of subdirectory being a file system identifier (fsid) of the flexible volume. Each fsid subdirectory (flexible volume) contains at least two files, a filesystem file and a storage label file. The storage label file is illustratively a 4kB file that contains meta-data similar to that stored in a conventional raid label. In other words, the storage label file is the analog of a raid label and, as such, contains information about the state of the flexible volume such as, e.g., the name of the flexible volume, a universal unique identifier (uuid) and fsid of the flexible volume, whether it is online, being created or being destroyed, etc.

Fig. 10 is a schematic block diagram of an on-disk representation of an aggregate 1000. The storage operating system 300, e.g., the RAID system 380, assembles a physical volume of pvbns to create the aggregate 1000, with pvbns 1 and 2 comprising a "physical" volinfo block 1002 for the aggregate. The volinfo block 1002 contains block pointers to fsinfo blocks 1004, each of which may represent a snapshot of the aggregate. Each fsinfo block 1004 includes a block pointer to an inode file 1006 that contains inodes of a plurality of files, including an owner map 1010, an active map 1012, a summary map 1014 and a space map 1016, as well as other special meta-data files. The inode file 1006 further includes a root directory 1020 and a "hidden" meta-data root directory 1030, the latter of which includes a namespace having files related to a flexible volume in which

users cannot "see" the files. The hidden meta-data root directory includes the *WAFL/fsid/* directory structure that contains filesystem file 1040 and storage label file 1090. Note that root directory 1020 in the aggregate is empty; all files related to the aggregate are organized within the hidden meta-data root directory 1030.

In addition to being embodied as a container file having level 1 blocks organized as a container map, the filesystem file 1040 includes block pointers that reference various file systems embodied as flexible volumes 1050. The aggregate 1000 maintains these flexible volumes 1050 at special reserved inode numbers. Each flexible volume 1050 also has special reserved inode numbers within its flexible volume space that are used for, among other things, the block allocation bitmap structures. As noted, the block allocation bitmap structures, e.g., active map 1062, summary map 1064 and space map 1066, are located in each flexible volume.

Specifically, each flexible volume 1050 has the same inode file structure/content as the aggregate, with the exception that there is no owner map and no *WAFL/fsid/filesystem file, storage label file* directory structure in a hidden meta-data root directory 1080. To that end, each flexible volume 1050 has a volinfo block 1052 that points to one or more fsinfo blocks 1054, each of which may represent a snapshot, along with the active file system of the flexible volume. Each fsinfo block, in turn, points to an inode file 1060 that, as noted, has the same inode structure/content as the aggregate with the exceptions noted above. Each flexible volume 1050 has its own inode file 1060 and distinct inode space with corresponding inode numbers, as well as its own root (fsid) directory 1070 and subdirectories of files that can be exported separately from other flexible volumes.

The storage label file 1090 contained within the hidden meta-data root directory 1030 of the aggregate is a small file that functions as an analog to a conventional raid label. A raid label includes physical information about the storage system, such as the volume name; that information is loaded into the storage label file 1090. Illustratively, the storage label file 1090 includes the name 1092 of the associated flexible volume 1050, the online/offline status 1094 of the flexible volume, and other identity and state informa-

tion 1096 of the associated flexible volume (whether it is in the process of being created or destroyed).

F. VLDB

Fig. 11 is a schematic block diagram illustrating a collection of management processes that execute as user mode applications 1100 on the storage operating system 300 to provide management of configuration information (i.e. management data) for the nodes of the cluster. To that end, the management processes include a management framework process 1110 and a volume location database (VLDB) process 1130, each utilizing a data replication service (RDB 1150) linked as a library. The management framework 1110 provides a user to an administrator 1170 interface via a command line interface (CLI) and/or a web-based graphical user interface (GUI). The management framework is illustratively based on a conventional common interface model (CIM) object manager that provides the entity to which users/system administrators interact with a node 200 in order to manage the cluster 100.

The VLDB 1130 is a database process that tracks the locations of various storage components (e.g., SVSs, flexible volumes, aggregates, etc.) within the cluster 100 to thereby facilitate routing of requests throughout the cluster. In the illustrative embodiment, the N-blade 310 of each node accesses a configuration table 235 that maps the SVS ID 502 of a data container handle 500 to a D-blade 350 that "owns" (services) the data container within the cluster. The VLDB includes a plurality of entries which, in turn, provide the contents of entries in the configuration table 235; among other things, these VLDB entries keep track of the locations of the flexible volumes (hereinafter generally "volumes 910") and aggregates 900 within the cluster. Examples of such VLDB entries include a VLDB volume entry 1200 and a VLDB aggregate entry 1300.

Fig. 12 is a schematic block diagram of an exemplary VLDB volume entry 1200. The entry 1200 includes a volume ID field 1205, an aggregate ID field 1210 and, in alternate embodiments, additional fields 1215. The volume ID field 1205 contains an ID that identifies a volume 910 used in a volume location process. The aggregate ID field 1210 identifies the aggregate 900 containing the volume identified by the volume ID field 1205. Likewise, Fig. 13 is a schematic block diagram of an exemplary VLDB aggregate

entry 1300. The entry 1300 includes an aggregate ID field 1305, a D-blade ID field 1310 and, in alternate embodiments, additional fields 1315. The aggregate ID field 1305 contains an ID of a particular aggregate 900 in the cluster 100. The D-blade ID field 1310 contains an ID of the D-blade hosting the particular aggregate identified by the aggregate ID field 1305.

The VLDB illustratively implements a RPC interface, e.g., a Sun RPC interface, which allows the N-blade 310 to query the VLDB 1130. When encountering contents of a data container handle 500 that are not stored in its configuration table, the N-blade sends an RPC to the VLDB process. In response, the VLDB 1130 returns to the N-blade the appropriate mapping information, including an ID of the D-blade that owns the data container. The N-blade caches the information in its configuration table 235 and uses the D-blade ID to forward the incoming request to the appropriate data container. All functions and interactions between the N-blade 310 and D-blade 350 are coordinated on a cluster-wide basis through the collection of management processes and the RDB library user mode applications 1100.

To that end, the management processes have interfaces to (are closely coupled to) RDB 1150. The RDB comprises a library that provides a persistent object store (storing of objects) for the management data processed by the management processes. Notably, the RDB 1150 replicates and synchronizes the management data object store access across all nodes 200 of the cluster 100 to thereby ensure that the RDB database image is identical on all of the nodes 200. At system startup, each node 200 records the status/state of its interfaces and IP addresses (those IP addresses it "owns") into the RDB database.

G. Storage System Architecture

The present invention is related to a storage system architecture illustratively comprising two or more volumes 910 distributed across a plurality of nodes 200 of cluster 100. The volumes are organized as a SVS and configured to store content of data containers, such as files and luns, served by the cluster in response to multi-protocol data access requests issued by clients 180. Notably, the content of each data container is apportioned among the volumes of the SVS to thereby improve the efficiency of storage ser-

vice provided by the cluster. To facilitate a description and understanding of the present invention, data containers are hereinafter referred to generally as "files".

According to an aspect of the invention, the SVS comprises a meta-data volume (MDV) and one or more data volumes (DV). The MDV is configured to store a canonical copy of meta-data, including access control lists (ACLs) and directories, associated with all files stored on the SVS, whereas each DV is configured to store, at least, data content of those files. For each file stored on the SVS, one volume is designated the CAV and, to that end, is configured to store ("cache") certain, rapidly-changing attribute meta-data associated with that file to thereby offload access requests that would otherwise be directed to the MDV. In the illustrative embodiment described herein, determination of the CAV for a file is based on a simple rule: designate the volume holding the first stripe of content (data) for the file as the CAV for the file. Not only is this simple rule convenient, but it also provides an optimization for small files. That is, a CAV may be able to perform certain operations without having to communicate with other volumes of the SVS if the file is small enough to fit within the specified stripe width. Ideally, the first stripes of data for files are distributed among the DVs of the SVS to thereby facilitate even distribution of CAV designations among the volumes of the SVS. In an alternate embodiment, data for files is striped across the MDV and the DVs.

Fig. 14 is a schematic block diagram of the inode files of an SVS 1400 in accordance with an embodiment of the present invention. The SVS 1400 illustratively comprises three volumes, namely MDV 1405 and two DVs 1410, 1415. It should be noted that in alternate embodiments additional and/or differing numbers of volumes may be utilized in accordance with the present invention. Illustratively, the MDV 1405 stores a plurality of inodes, including a root directory (RD) inode 1420, a directory (DIR) inode 1430, file (F) inodes 1425, 1435, 1445 and an ACL inode 1440. Each of these inodes illustratively includes meta-data (M) associated with the inode. In the illustrative embodiment, each inode on the MDV 1405 does not include data (D); however, in alternate embodiments, the MDV may include user data.

In contrast, each DV 1410, 1415 stores only file (F) inodes 1425, 1435, 1445 and ACL inode 1440. According to the inventive architecture, a DV does not store directo-

ries or other device inodes/constructs, such as symbolic links; however, each DV does store F inodes, and may store cached copies of ACL inodes, that are arranged in the same locations as their respective inodes in the MDV 1405. A particular DV may not store a copy of an inode until an I/O request for the data container associated with the inode is received by the D-Blade serving a particular DV. Moreover, the contents of the files denoted by these F inodes are periodically sparse according to SVS striping rules, as described further herein. In addition, since one volume is designated the CAV for each file stored on the SVS 1400, DV 1415 is designated the CAV for the file represented by inode 1425 and DV 1410 is the CAV for the files identified by inodes 1435, 1445. Accordingly, these CAVs cache certain, rapidly-changing attribute meta-data (M) associated with those files such as, e.g., file size 615, as well as access and/or modification time stamps 620.

According to another aspect of the invention, the SVS is associated with a set of striping rules that define a stripe algorithm, a stripe width and an ordered list of volumes within the SVS. The striping rules for each SVS are illustratively stored as an entry of VLDB 1130 and accessed by SVS ID. Fig. 15 is a schematic block diagram of an exemplary VLDB SVS entry 1500 in accordance with an embodiment of the present invention. The VLDB entry 1500 includes a SVS ID field 1505 and one or more sets of striping rules 1530. In alternate embodiments additional fields 1535 may be included. The SVS ID field 1505 contains the ID of a SVS which, in operation, is specified in data container handle 500.

Each set of striping rules 1530 illustratively includes a stripe width field 1510, a stripe algorithm ID field 1515, an ordered list of volumes field 1520 and, in alternate embodiments, additional fields 1525. The striping rules 1530 contain information for identifying the organization of a SVS. For example, the stripe algorithm ID field 1515 identifies a striping algorithm used with the SVS. In the illustrative embodiment, multiple striping algorithms could be used with a SVS; accordingly, stripe algorithm ID is needed to identify which particular algorithm is utilized. Each striping algorithm, in turn, specifies the manner in which file content is apportioned as stripes across the plurality of volumes of the SVS. The stripe width field 1510 specifies the size/width of each stripe. The ordered list of volumes field 1520 contains the IDs of the volumes comprising the SVS.

In an illustrative embodiment, the ordered list of volumes comprises a plurality of tuples comprising of a flexible volume ID and the aggregate ID storing the flexible volume. Moreover, the ordered list of volumes may specify the function and implementation of the various volumes and striping rules of the SVS. For example, the first volume in the ordered list may denote the MDV of the SVS, whereas the ordering of volumes in the list may denote the manner of implementing a particular striping algorithm, e.g., round-robin.

A Locate( ) function 375 is provided that enables the VSM 370 and other modules (such as those of N-blade 310) to locate a D-blade 350 and its associated volume of a SVS 1400 in order to service an access request to a file. The Locate() function takes as arguments, at least (i) a SVS ID 1505, (ii) an offset within the file, (iii) the inode number for the file and (iv) a set of striping rules 1530, and returns the volume 910 on which that offset begins within the SVS 1400. For example, assume a data access request directed to a file is issued by a client 180 and received at the N-blade 310 of a node 200, where it is parsed through the multi-protocol engine 325 to the appropriate protocol server of N-blade 310.

To determine the location of a D-blade 350 to which to transmit a CF message 400, the N-blade 310 may first retrieve a SVS entry 1500 to acquire the striping rules 1530 (and list of volumes 1520) associated with the SVS. The N-blade 310 then executes the Locate() function 375 to identify the appropriate volume to which to direct an operation. Thereafter, the N-Blade may retrieve the appropriate VLDB volume entry 1200 to identify the aggregate containing the volume and the appropriate VLDB aggregate entry 1300 to ultimately identify the appropriate D-blade 350. The protocol server of N-blade 310 then transmits the CF message 400 to the D-blade 350.

Fig. 16 is a schematic block diagram illustrating the periodic sparseness of file content stored on volumes A 1605, B 1610 and C 1615 of SVS 1600 in accordance with an embodiment of the present invention. As noted, file content is periodically sparse according to the SVS striping rules, which specify a striping algorithm (as indicated by stripe algorithm ID field 1515) and a size/width of each stripe (as indicated by stripe width field 1510). Note that, in the illustrative embodiment, a stripe width is selected to

ensure that each stripe may accommodate the actual data (e.g., stored in data blocks 806) referenced by an indirect block (e.g., level 1 block 804) of a file.

In accordance with an illustrative round robin striping algorithm, volume A 1605 contains a stripe of file content or data (D) 1620 followed, in sequence, by two stripes of sparseness (S) 1622, 1624, another stripe of data (D) 1626 and two stripes of sparseness (S) 1628, 1630. Volume B 1610, on the other hand, contains a stripe of sparseness (S) 1632 followed, in sequence, by a stripe of data (D) 1634, two stripes of sparseness (S) 1636, 1638, another stripe of data (D) 1640 and a stripe of sparseness (S) 1642. Volume C 1615 continues the round robin striping pattern and, to that end, contains two stripes of sparseness (S) 1644, 1646 followed, in sequence, by a stripe of data (D) 1648, two stripes of sparseness (S) 1650, 1652 and another stripe of data (D) 1654.

H. Re-striping Operations

The present invention is directed to a system and method for re-striping one or more data containers across a SVS that has been modified by the addition/removal of one or more volumes. Illustratively, an administrator creates additional volumes using, e.g., conventional volume create commands available via the user interface of management framework 1110. The newly-created additional volumes may be configured for service by one or more D-blades 350 of the cluster 100; however, to achieve maximum load balancing, each additional volume is preferably serviced by a separate D-blade.

Fig. 17 is a flow chart detailing the steps of a procedure 1700 for performing a re-striping operation in accordance with a first embodiment of the present invention. The procedure 1700 begins in step 1705 and continues to step 1710 where an administrator creates one or more additional volumes for use with a SVS. In step 1715, a new set of striping rules (new rule set) 1530 is created, e.g., in response to a re-striping operation request received by an N-blade 310 from the administrator. Creation of the new rule set is illustratively effected via user interface operations that specify, in this embodiment, any changes to the specific rules of the existing rule set. For example, although the stripe algorithm 1515 and stripe width 1510 may remain the same, the new rule set incorporates the additional volumes as constituents of the ordered list of volumes 1520 in the SVS. In step 1720, the new rule set is marked as "new" and, in step 1725, installed in a SVS entry

1500 of the VLDB 1130. As described herein, any subsequent data access request received at an N-blade 310 of a node 200 is processed in accordance with the new striping rules. In addition, the existing rule set in the SVS entry 1500 is marked "old" (step 1730).

In step 1735, the N-blade initiates the re-striping operation by instructing each D-blade of each node about the re-striping request. As noted, the VSM 370 of each D-blade 350 is configured to implement the Locate() function 375 to compute the location of data container content (i.e., a stripe) in its SVS volume. In step 1740, the VSM determines whether the volume holds a stripe of a file affected by the re-striping operation. If not, the procedure completes at step 1755. Otherwise, the VSM re-locates the content of the stripe to the additional volume in accordance with the new striping rules (step 1745). As described further herein, data relocation illustratively occurs as a background procedure to ensure that D-blades prioritize servicing of subsequent data access requests issued by clients. Upon completion of data relocation among the additional volumes, the old set of striping rules is deleted from the SVS entry of the VLDB in step 1750 and the procedure ends at Step 1755.

Fig. 18 is a schematic block diagram of an exemplary SVS 1800 after the addition of a volume (volume D) 1805 to the striped volume set 1600 of Fig. 16. Initially, after creation of volume 1805 and prior to re-striping, the entire contents of that volume reflect stripes of sparseness (S). However, after completion of the re-striping procedure of Fig. 17, the contents of volume 1805 are modified according to a specific striping algorithm. Fig. 19 is a schematic block diagram of an exemplary SVS 1900 after completion of the re-striping procedure of Fig. 17. Assuming a round robin striping algorithm, volume D 1805 has been fully integrated into the SVS and, as such, contains a stripe of data (D) 1814 in round-robin relation to the other stripes of data (D) across the volumes of SVS 1900. Specifically, volume D 1805 contains three stripes of sparseness (S) 1808, 1810, 1812, followed, in sequence, by one stripe of data (D) 1814 and two stripes of sparseness (S) 1816, 1818. It should be noted that in other embodiments, other striping algorithms may be utilized which will result in different striping patterns and sequences once additional volume(s) have been integrated into the SVS.

As noted, any data access request received at an N-blade 310 of a node 200 subsequent to the initiation of the re-striping procedure 1700 is processed in accordance with the new striping rules. However, if a subsequent data access request is forwarded to a D-blade 350 of an additional volume that has yet to receive the re-located content of a stripe according to the new striping rules, the VSM 370 of the D-blade serving that volume uses the old set of striping rules to retrieve the data from the volume currently storing the stripe so that it may promptly service the request. Fig. 20 is a flowchart detailing the steps of a procedure 2000 for processing a data access request directed to a volume of a SVS prior to completion of a re-striping operation in accordance with an embodiment of the present invention. The procedure begins in step 2005 and continues to step 2010 where a VSM of a D-blade receives the request directed to its volume of the SVS. In step 2015, the VSM determines whether the stripe to which the request is directed is present on the volume. If so, the VSM, in cooperation with the file system 360, processes the request according to conventional techniques in step 2035. An example of a conventional technique for processing data access requests directed to a SVS is described in the above-referenced U.S. Patent Application entitled STORAGE SYSTEM ARCHITECTURE FOR STRIPING DATA CONTAINER CONTENT ACROSS VOLUMES OF A CLUSTER. The procedure then completes in step 2040.

However, if the stripe is not present on the volume (thus denoting an additional volume that has yet to receive the re-located content on the stripe), the procedure branches to step 2020 where the VSM retrieves the content of the requested stripe from the D-blade of the volume that currently stores the stripe. To that end, the VSM implements the Locate() function 375 to compute the location of the stripe (data container content) using the old set of striping rules maintained in the SVS entry 1500. Such inter-D-blade data retrieval may be performed by generating appropriate CF messages 400 transferred over the cluster switching fabric 150. Upon retrieving the content of the requested stripe, the VSM 370 writes the content (data) to the appropriate stripe on its volume (step 2025). In step 2030, the VSM cooperates with the file system to process the request and the procedure completes in step 2040.

As can be appreciated, the procedure 2000 operates to ensure that stripes are re-located to their proper locations in response to data access requests directed to the stripes.

However, the content of certain stripes may not be accessed for substantial periods of time, resulting in a slow re-striping procedure. To expedite re-striping, data relocation illustratively occurs as a background procedure to constantly, but non-disruptively, move file content in accordance with the new set of striping rules. Fig. 21 is a flow chart detailing the steps of a background procedure 2100 for re-locating data in accordance with an embodiment of the present invention. The procedure 2100 begins in step 2105 and continues to step 2110 where the VSM selects a non-sparse stripe within a volume serviced by its D-blade. In step 2115, the VSM determines whether the selected stripe belongs on the volume according to the new set of striping rules using, e.g., the Locate() function. If so, the procedure branches to step 2135.

However, if the selected stripe does not belong the volume, the VSM determines on which volume the stripe should reside (step 2120) using, e.g., the Locate() function in connection with the new set of striping rules. In step 2125, the VSM forwards the content of the stripe to the D-blade serving the appropriate volume and the stripe content is stored at the appropriate location. In step 2130, the VSM deletes its local copy of the stripe from its volume to ensure the volume remains periodically sparse. In step 2135, the VSM determines whether there are additional stripes to be checked. If so, the procedure returns to step 2110; otherwise, the procedure completes in step 2140.

In a second embodiment of the invention, a SVS 1400 is associated with multiple (e.g., new and old) sets of striping rules, each defining a stripe algorithm, a stripe width and an ordered list of volumes distributed across a plurality of nodes in the cluster. Here, the new set of striping rules is used for any newly created data containers (files) stored on the SVS, while the old set of striping rules is used for all existing files stored on the SVS. For example, a fourth volume (volume 1805) added to a three volume SVS (SVS 1800) is not used in the SVS until such time as a new file is created. Yet once the new file is created, the content (data) of that file is striped across all four volumes in accordance with the new striping rules. This technique may be expanded by, for example, adding a fifth volume and yet another set of striping rules such that files may be striped across three, four or five volumes of the SVS. Essentially, each file in the SVS is associated with a set of striping rules. This second embodiment of the invention advantageously enables substantially instantaneous re-striping of new files without disrupting the existing files.

According to an aspect of the invention, the association between a data container (file) and set of striping rules is illustratively effected through the use of an additional field added to the data container handle 500 used to access the file. The data container handle is convenient for rendering such an association because it is accessible by the N-blade 310, which needs to know the appropriate set of striping rules for the file when determining to which D-blade 350 (and volume) to re-direct a data access request. Fig. 22 is a schematic block diagram of the format of a data container handle 2200 that is modified to include an additional field 2210 configured to store an identifier (ID) of a striping rule set. Note that the field 2210 may illustratively replace the unique-ifier field 506.

Alternatively, a generation value 2220 of the data container handle can be manipulated (i.e., "overloaded") to identify the set of striping rules to be applied to the data container. Here, each striping rule set is associated with a particular range of generation values. For example, a first rule set may be associated with generation values 0-1,000,000 and a second rule set may be associated with generation values of 1,000,001 – 2,000,000, etc. For this alternative embodiment, there may be no need to modify the data container handle 500, as each handle typically includes a generation value 2200. When creating a new set of striping rules, a new "floor" may be placed on the generation value, so that a file (inode) allocated after the creation of the new rule set will have a generation value 2200 sufficiently large to require use of the new-created set of striping rules. This generation value floor is illustratively stored in a generation value floor field 1517 of the striping rules 1530.

Fig. 23 is a flow chart detailing the steps of a procedure 2300 for performing a re-striping operation in accordance with a second embodiment of the present invention. The procedure 2300 begins in step 2305 and continues to step 2310 where an N-blade 310 receives a data access request directed to a data container (file) of a SVS. In step 2315, the N-blade determines which volume of the SVS stores the desired stripe of the file. In the embodiment where the data container handle is modified to include the rule set ID 2210, the N-blade invokes the Locate() function 375 using the rule set ID and appropriate offset to identify the volume. In step 2320, the N-blade forwards the request to the D-blade 350 serving the volume and, in step 2325, the VSM 370 of the D-blade cooperates

with the file system 360 to process the request. The procedure then completes in step 2330.

Fig. 24 is a schematic block diagram of an exemplary SVS 2400 after the re-striping operation in accordance with the second embodiment of the present invention. Assume that for each set of striping rules, the striping algorithm remains the same, e.g., a round-robin algorithm. Initially, content for a first file, File A ($F_A$), is striped across three original volumes 2402, 2404 and 2406 of SVS 2400 according to a first set of striping rules. As a result, $F_A$ occupies stripes 2412, 2430, 2448, 2418, 2436, 2454, 2424 and 2442 of SVS 2400. At some later point in time, a fourth volume 2408 is added to the SVS 2400 and a second file, File B ($F_B$ ), is striped across the four volumes 2402-2408 using a new set of striping rules. $F_B$ consequently occupies stripes 2414, 2432, 2450, 2468, 2420, 2438, 2456 and 2474 of the SVS. Notably, although $F_B$ has been striped across all four volumes of the SVS, $F_A$ remains striped across the original three volumes. At yet another later point in time, a fifth volume 2410 is added to the SVS 2400 and a third file, File C ($F_C$), is striped across the five volumes 2402-2410 using yet another new set of striping rules. Therefore, $F_C$ occupies stripes 2416, 2434, 2452, 2470, 2488, 2422, 2440 and 2458. It is thus apparent that this second re-striping embodiment does not guarantee full re-striping of all files across all volumes in a SVS until each "older" file is deleted.

The foregoing description has been directed to particular embodiments of this invention. It will be apparent, however, that other variations and modifications may be made to the described embodiments, with the attainment of some or all of their advantages. Specifically, it should be noted that the principles of the present invention may be implemented in non-distributed file systems. Furthermore, while this description has been written in terms of N and D-blades, the teachings of the present invention are equally suitable to systems where the functionality of the N and D-blades are implemented in a single system. Alternately, the functions of the N and D-blades may be distributed among any number of separate systems, wherein each system performs one or more of the functions. Additionally, the procedures, processes and/or modules described herein may be implemented in hardware, software, embodied as a computer-readable medium having program instructions, firmware, or a combination thereof. Therefore, it is

the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the invention.

What is claimed is:

# CLAIMS

1. A method for restriping data across a striped volume set, the method comprising the steps of:

creating a new striping rule set identifying a set of volumes to be utilized as the striped volume set;

in response to receiving an operation directed to a stripe of the data, determining if the stripe is located on a correct volume of the set of volumes; and

in response to determining that the stripe is not located on the correct volume, relocating the stripe to the correct volume before processing the received operation.

2. The method of claim 1 further comprising the steps of:

for each stripe in the striped volume set, determining if the stripe is located on a correct volume;

in response to determining that the stripe is not located on the correct volume, relocating the stripe to the correct volume.

3. The method of claim 1 wherein the step of relocating the stripe comprises the step of copying the stripe from a current volume to the correct volume.

4. The method of claim 1 wherein the step of determining if the stripe is located on the correct volume of the set of volumes comprises the step of using a locate function with the new striping rule set to determine the correct volume.

5. A system for restriping data across a striped volume set, the system comprising:

means for creating a new striping rule set identifying a set of volumes to be utilized as the striped volume set;

means for determining, in response to receiving an operation directed to a stripe of the data, if the stripe is located on a correct volume of the set of volumes; and

means for relocating, in response to determining that the stripe is not located on the correct volume, the stripe to the correct volume before processing the received operation.

6. The system of claim 5 further comprising:

means for determining, for each stripe in the striped volume set, if the stripe is located on a correct volume; and

means for relocating, in response to determining that the stripe is not located on the correct volume, the stripe to the correct volume.

7. The system of claim 5 wherein the means for relocating the stripe comprises means for copying the stripe from a current volume to the correct volume.

8. The system of claim 5 wherein the means for determining if the stripe is located on the correct volume of the set of volumes comprises means for using a locate function with the new striping rule set to determine the correct volume.

9. A computer readable medium for restriping data across a striped volume set, the computer readable medium including program instructions for performing the steps of::

creating a new striping rule set identifying a set of volumes to be utilized as the striped volume set;

in response to receiving an operation directed to a stripe of the data, determining if the stripe is located on a correct volume of the set of volumes; and

in response to determining that the stripe is not located on the correct volume, relocating the stripe to the correct volume before processing the received operation.

10. A system for restriping data across a striped volume set, the system comprising:

a plurality of computers, each of the plurality of computers comprising a container striping module, each container striping module adapted to determine, for each stripe in the striped volume set, if the stripe is currently located on a correct volume and,

in response to determining that the stripe is not on a correct volume, relocating the stripe to the correct volume.

11. The system of claim 10 wherein the determination of whether the stripe is on the correct volume is made by a locate function that analyzes an inode, an offset and a set of striping rules.

12. The system of claim 11 wherein the set of striping rules comprises an ordered list of volumes.

13. A method for restriping data across a striped volume set, the method comprising the steps of:

receiving a command directed to a file in the striped volume set;

determining which of a plurality of volumes of the striped volume set stores a stripe associated with the received command, the determination being made by selecting one of a plurality of striping rules associated with the striped volume set; and

forwarding the received command to a computer serving the volume storing the stripe associated with the command.

14. The method of claim 13 wherein the step of selecting one of a plurality of file striping rules associated with the striped volume set comprises the step of analyzing a striped volume set identifier in a file handle associated with the received command.

15. The method of claim 13 wherein the step of selecting one of a plurality of striping rules associated with the striped volume set comprises the step of analyzing an inode generation value associated with an inode associated with a data container being operated on by the received command.

16. The method of claim 15 wherein each of the plurality of striping rules comprises a generation value floor that identifies which inodes are associated each of the plurality of striping rules.

17. A system for restriping data across a striped volume set, the system comprising:

means for receiving a command directed to a file in the striped volume set;

means for determining which of a plurality of volumes of the striped volume set stores a stripe associated with the received command, the determination being made by selecting one of a plurality of striping rules associated with the striped volume set; and

means for forwarding the received command to a computer serving the volume storing the stripe associated with the command.

18. The system of claim 17 wherein the means for selecting one of a plurality of striping rules associated with the striped volume set comprises means for analyzing a striped volume set identifier in a file handle associated with the received command.

19. The method of claim 17 wherein the means for selecting one of a plurality of striping rules associated with the striped volume set comprises means for analyzing an inode generation value associated with an inode associated with a data container being operated on by the received command.

20. A computer readable medium for restriping data across a striped volume set, the computer readable medium including program instructions for performing the steps of:

receiving a command directed to a file in the striped volume set;

determining which of a plurality of volumes of the striped volume set stores a stripe associated with the received command, the determination being made by selecting one of a plurality of striping rules associated with the striped volume set; and

forwarding the received command to a computer serving the volume storing the stripe associated with the command.

21. A system for restriping data across a striped volume set, the system comprising:

one or more storage servers having a container striping module adapted to process commands directed to the striped volume set; and

at least one multi-protocol engine having a cluster fabric interface module adapted to determine which of a plurality of striping rules to utilize in processing a received command, the cluster fabric interface module further adapted to forward the received command to one of the one or more storage servers for processing in response to determining which of the plurality of striping rules to utilize.

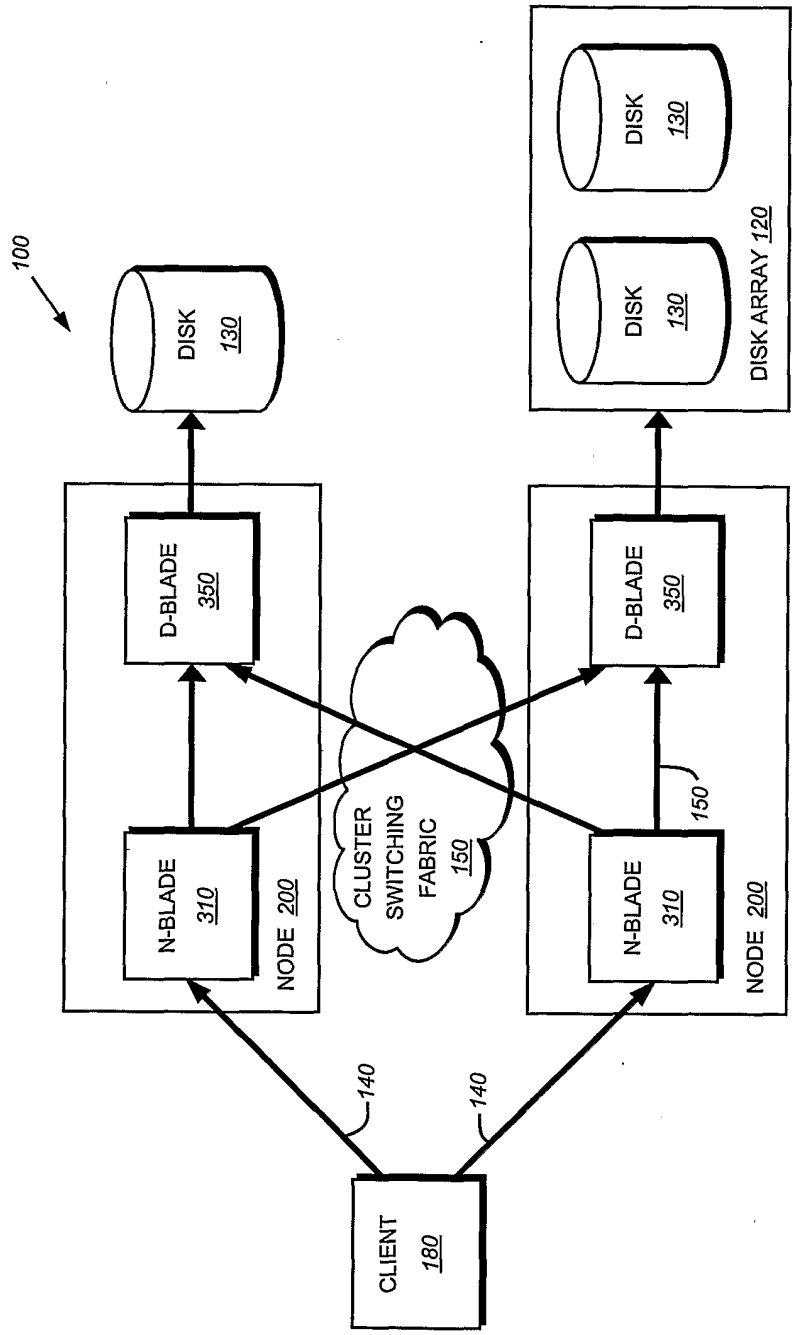

100
DISK 130
DISK 130
DISK 130
DISK ARRAY 120
D-BLADE 350
D-BLADE 350
N-BLADE 310
N-BLADE 310
NODE 200
NODE 200
CLUSTER SWITCHING FABRIC 150
150
140
140
CLIENT 180

FIG. 1

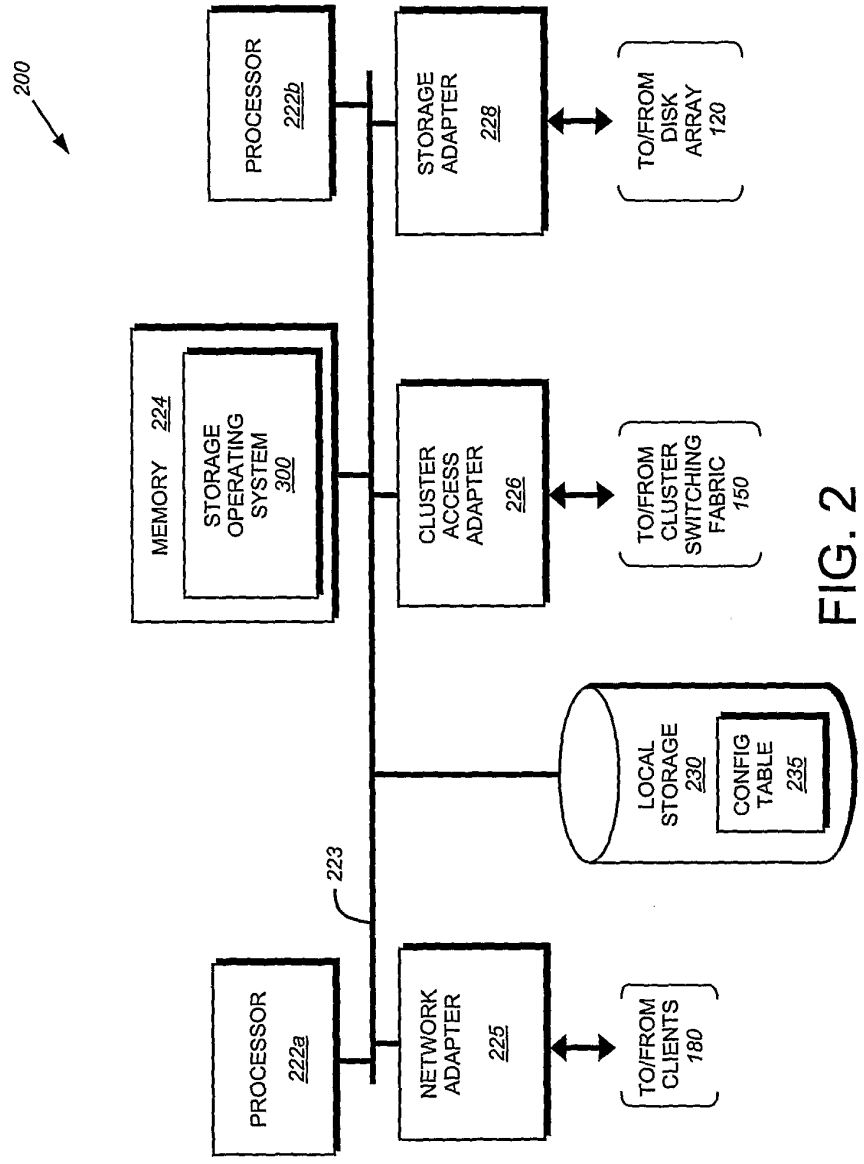

200
PROCESSOR 222b
STORAGE ADAPTER 228
TO/FROM DISK ARRAY 120
MEMORY 224
STORAGE OPERATING SYSTEM 300
CLUSTER ACCESS ADAPTER 226
TO/FROM CLUSTER SWITCHING FABRIC 150
LOCAL STORAGE 230
CONFIG TABLE 235
223
PROCESSOR 222a
NETWORK ADAPTER 225
TO/FROM CLIENTS 180

FIG. 2

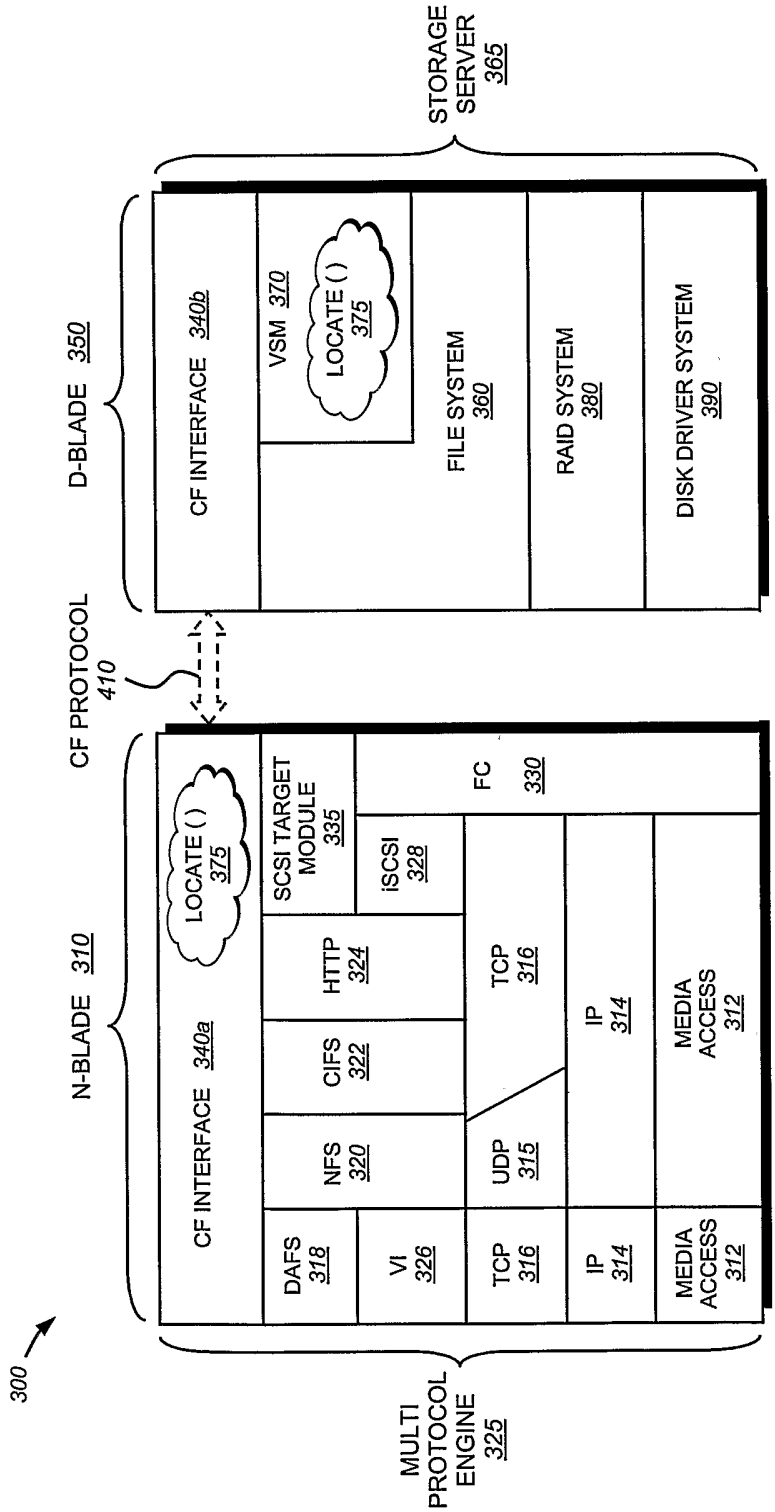

300
MULTI PROTOCOL ENGINE 325
N-BLADE 310
CF PROTOCOL 410
D-BLADE 350
STORAGE SERVER 365
CF INTERFACE 340a
LOCATE ( ) 375
DAFS 318
NFS 320
CIFS 322
HTTP 324
VI 326
SCSI TARGET MODULE 335
iSCSI 328
FC 330
TCP 316
UDP 315
IP 314
MEDIA ACCESS 312
CF INTERFACE 340b
VSM 370
LOCATE ( ) 375
FILE SYSTEM 360
RAID SYSTEM 380
DISK DRIVER SYSTEM 390

FIG. 3

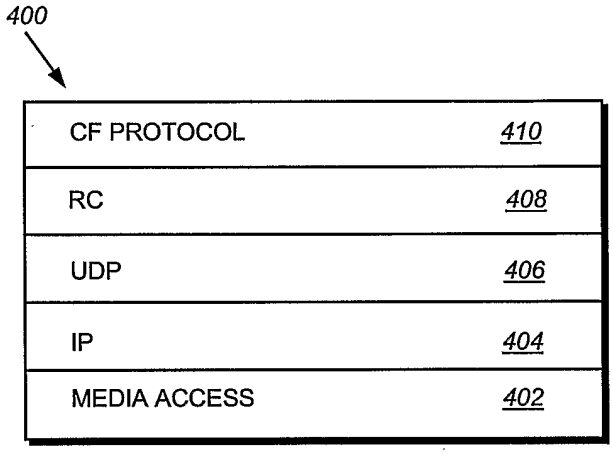

400
CF PROTOCOL 410
RC 408
UDP 406
IP 404
MEDIA ACCESS 402

FIG. 4

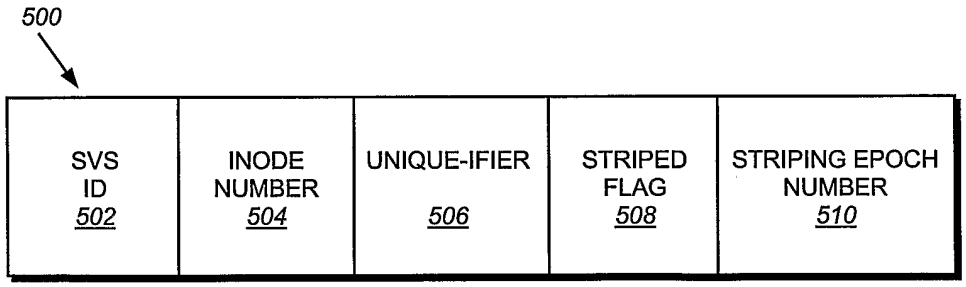

500
SVS ID 502
INODE NUMBER 504
UNIQUE-IFIER 506
STRIPED FLAG 508
STRIPING EPOCH NUMBER 510

FIG. 5

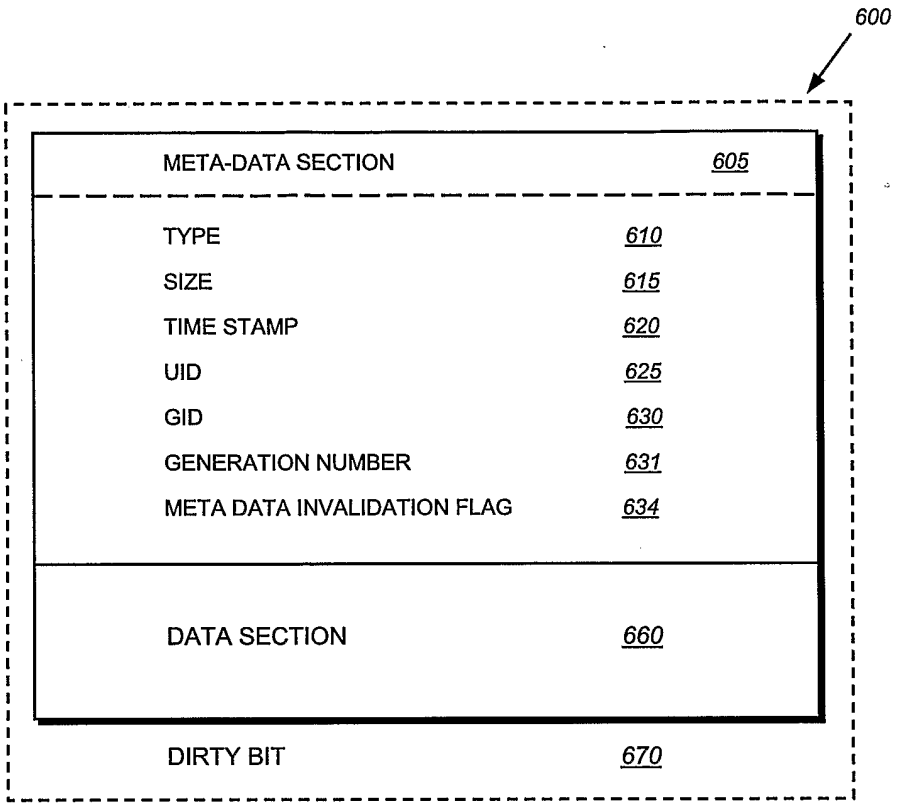

600
META-DATA SECTION
605
TYPE
610
SIZE
615
TIME STAMP
620
UID
625
GID
630
GENERATION NUMBER
631
META DATA INVALIDATION FLAG
634
DATA SECTION
660
DIRTY BIT
670

FIG. 6

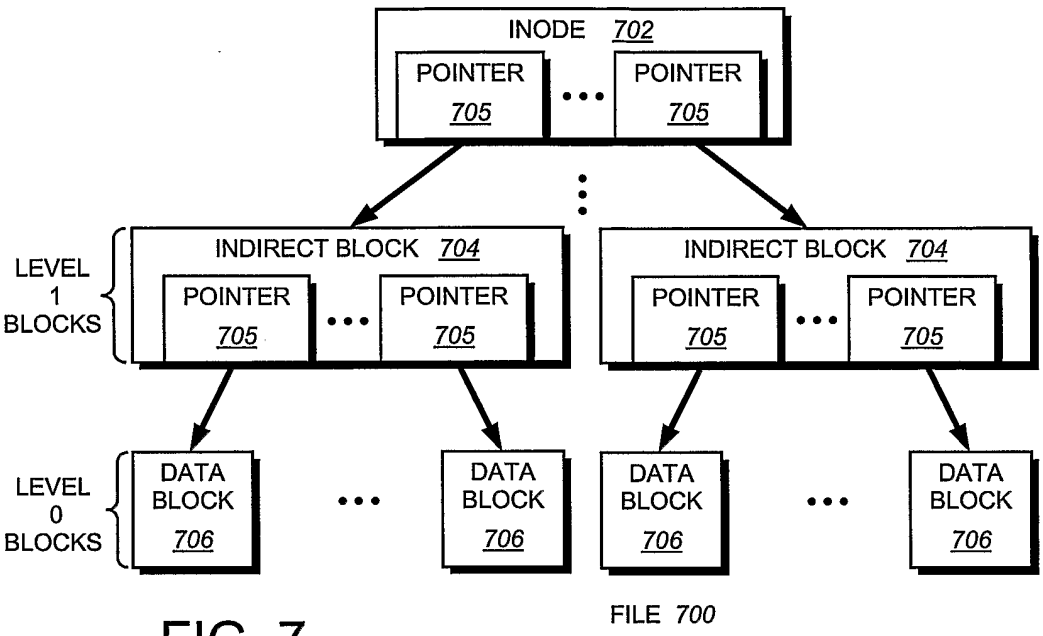

INODE 702
POINTER 705
POINTER 705
INDIRECT BLOCK 704
INDIRECT BLOCK 704
LEVEL 1 BLOCKS
POINTER 705
POINTER 705
POINTER 705
POINTER 705
LEVEL 0 BLOCKS
DATA BLOCK 706
DATA BLOCK 706
DATA BLOCK 706
DATA BLOCK 706
FILE 700

FIG. 7

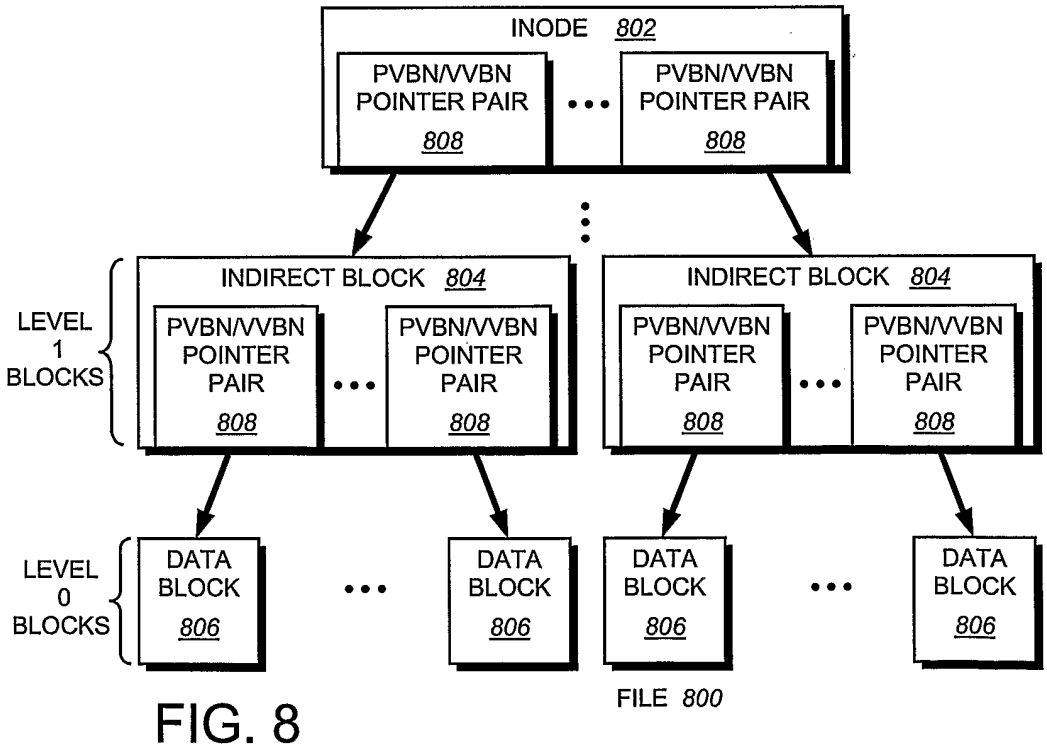

INODE 802
PVBN/VVBN POINTER PAIR 808
PVBN/VVBN POINTER PAIR 808
INDIRECT BLOCK 804
INDIRECT BLOCK 804
LEVEL 1 BLOCKS
PVBN/VVBN POINTER PAIR 808
PVBN/VVBN POINTER PAIR 808
PVBN/VVBN POINTER PAIR 808
PVBN/VVBN POINTER PAIR 808
LEVEL 0 BLOCKS
DATA BLOCK 806
DATA BLOCK 806
DATA BLOCK 806
DATA BLOCK 806
FILE 800

FIG. 8

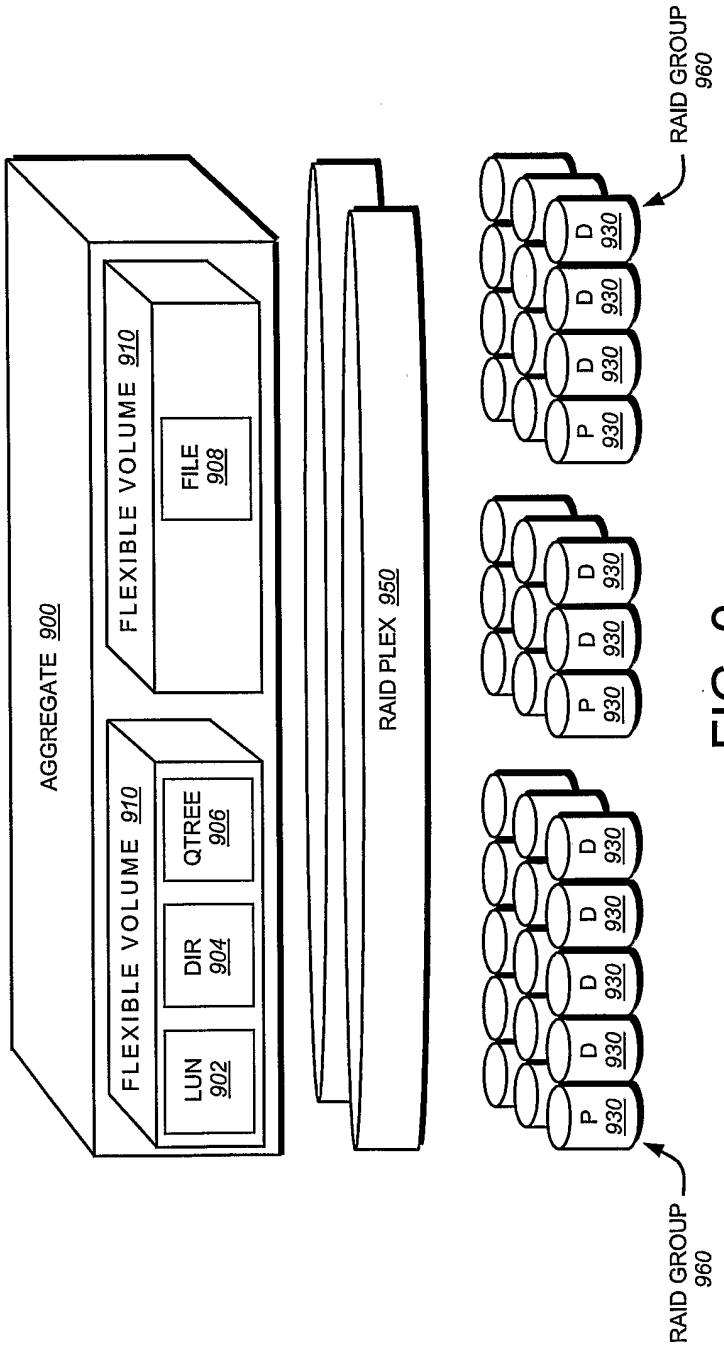

AGGREGATE 900
FLEXIBLE VOLUME 910
LUN 902
DIR 904
QTREE 906
FLEXIBLE VOLUME 910
FILE 908
RAID PLEX 950
RAID GROUP 960
RAID GROUP 960
P 930
D 930

FIG. 9

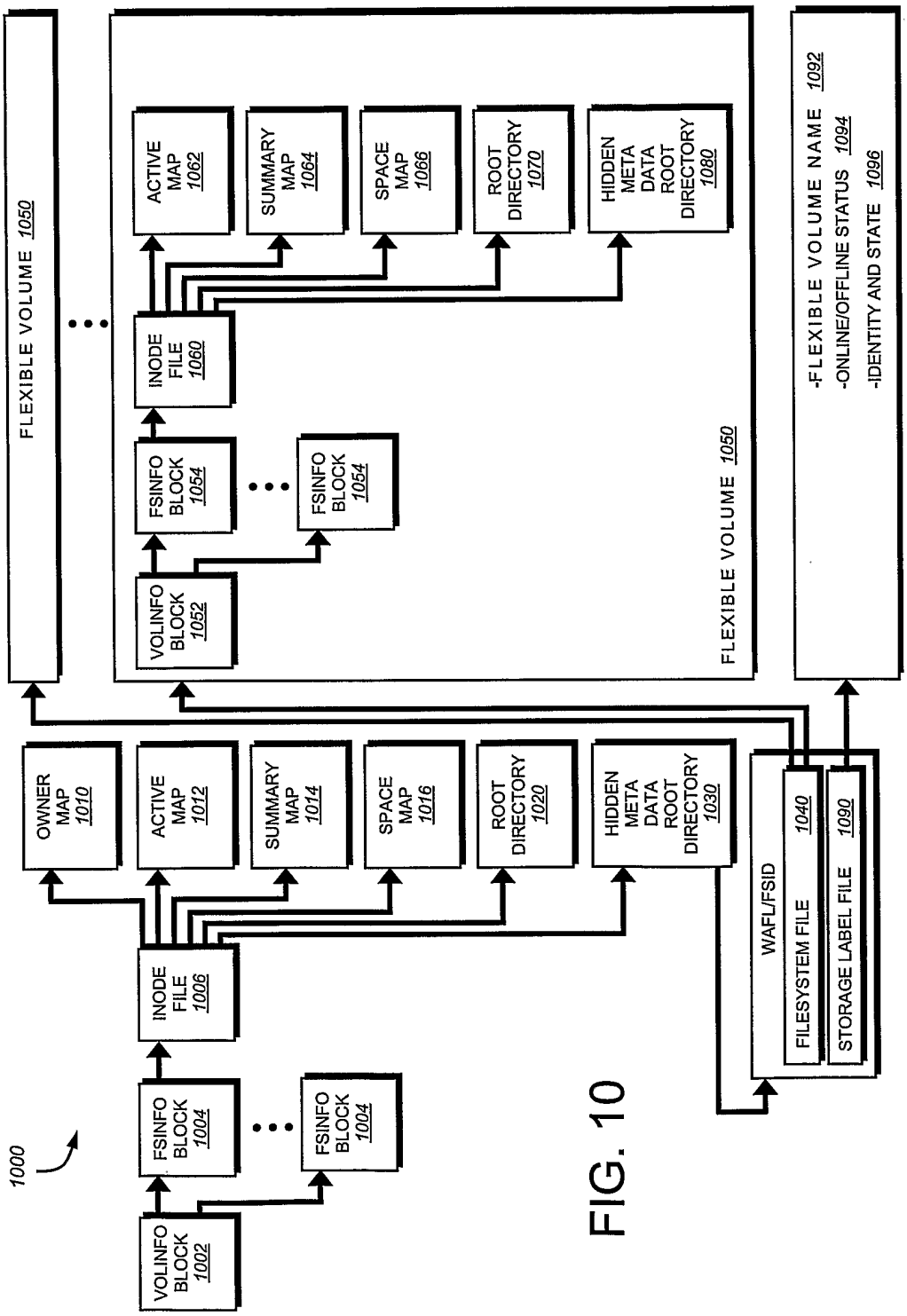

1000
VOLINFO BLOCK 1002
FSINFO BLOCK 1004
FSINFO BLOCK 1004
INODE FILE 1006
OWNER MAP 1010
ACTIVE MAP 1012
SUMMARY MAP 1014
SPACE MAP 1016
ROOT DIRECTORY 1020
HIDDEN META DATA ROOT DIRECTORY 1030
WAFL/FSID
FILESYSTEM FILE 1040
STORAGE LABEL FILE 1090
FLEXIBLE VOLUME 1050
VOLINFO BLOCK 1052
FSINFO BLOCK 1054
FSINFO BLOCK 1054
INODE FILE 1060
ACTIVE MAP 1062
SUMMARY MAP 1064
SPACE MAP 1066
ROOT DIRECTORY 1070
HIDDEN META DATA ROOT DIRECTORY 1080
FLEXIBLE VOLUME 1050
-FLEXIBLE VOLUME NAME 1092
-ONLINE/OFFLINE STATUS 1094
-IDENTITY AND STATE 1096

FIG. 10

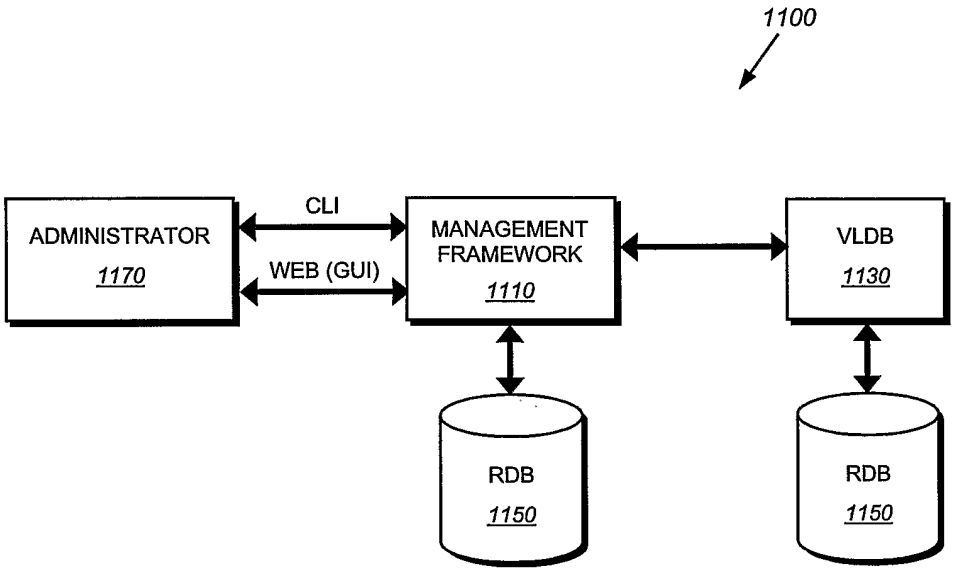

1100
ADMINISTRATOR
1170
CLI
WEB (GUI)
MANAGEMENT FRAMEWORK
1110
VLDB
1130
RDB
1150
RDB
1150

FIG. 11

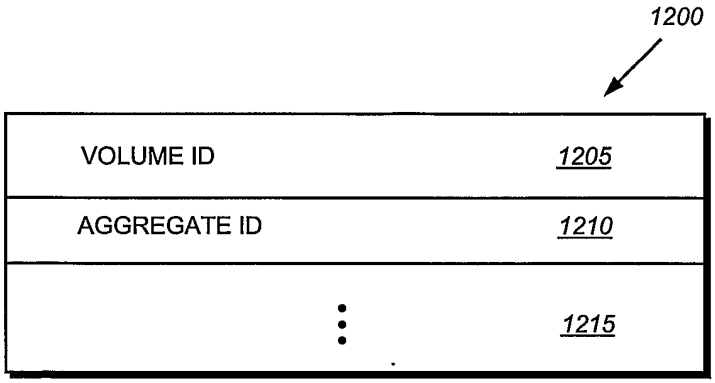

1200
VOLUME ID
1205
AGGREGATE ID
1210
1215

FIG. 12

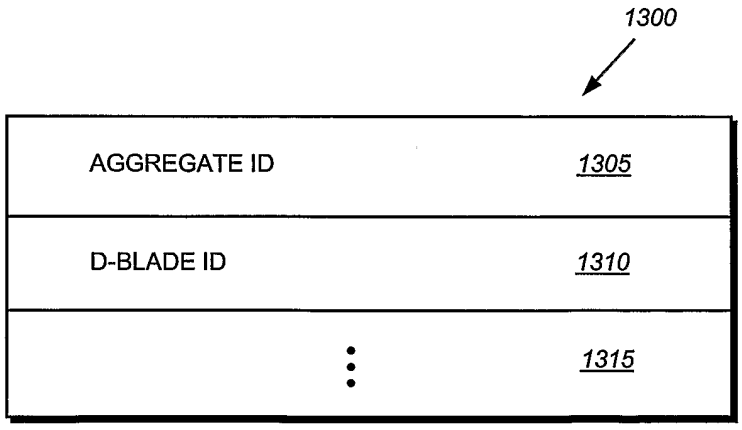

1300
AGGREGATE ID
1305
D-BLADE ID
1310
1315

FIG. 13

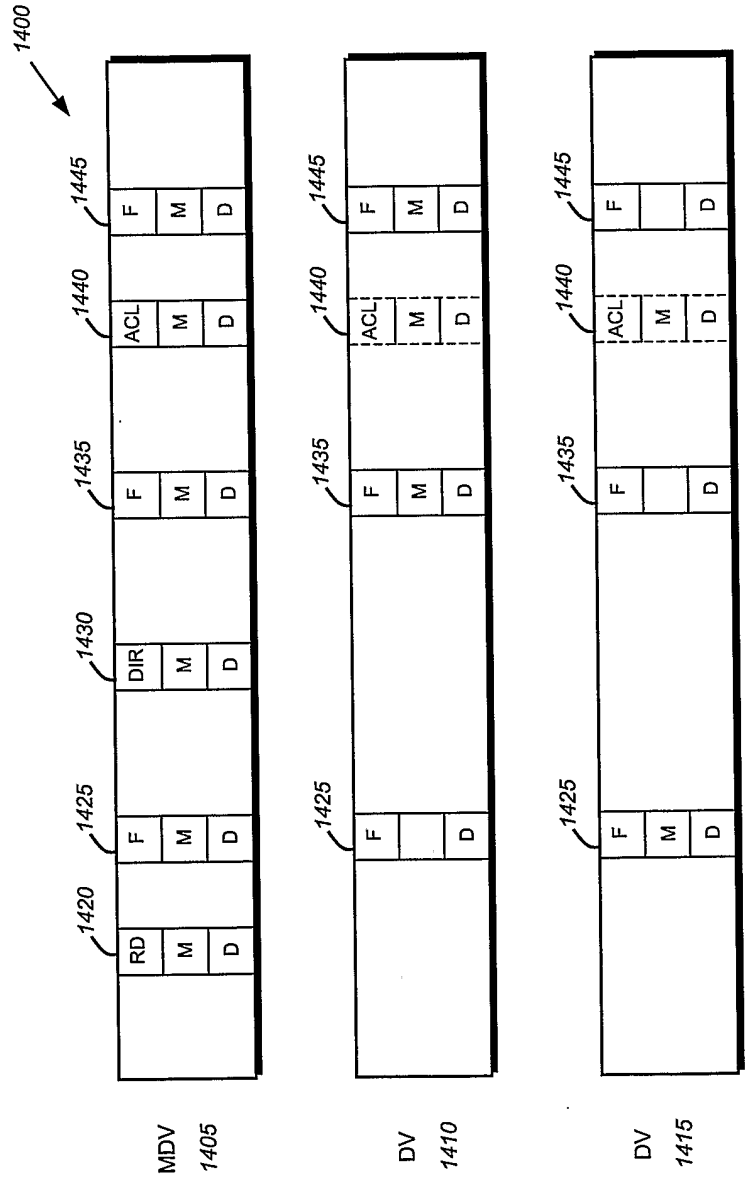

1400
1445
1440
1435
1430
1425
1420
F
M
D
ACL
DIR
RD
MDV
1405
DV
1410
DV
1415

FIG. 14

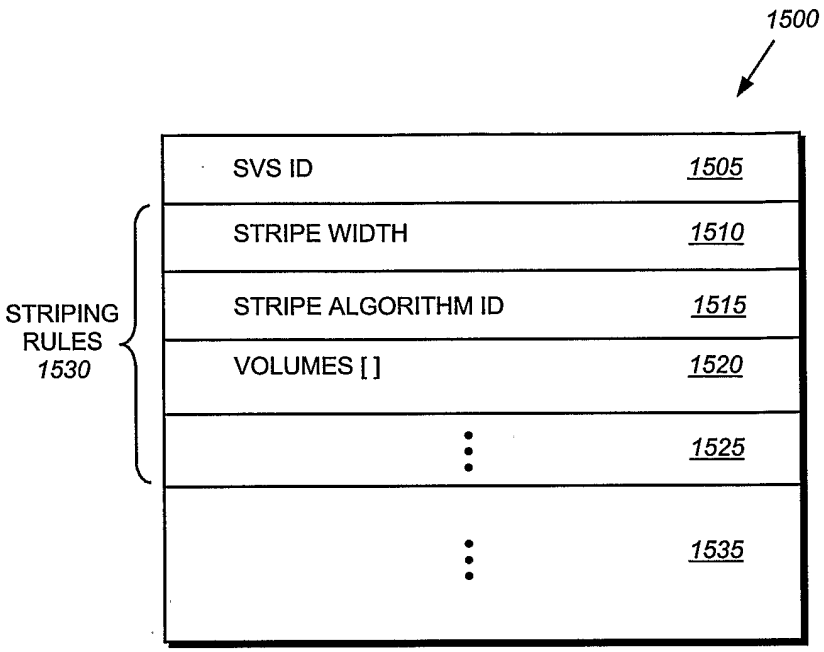

1500
SVS ID
1505
STRIPE WIDTH
1510
STRIPE ALGORITHM ID
1515
VOLUMES [ ]
1520
1525
1535
STRIPING RULES 1530

FIG. 15

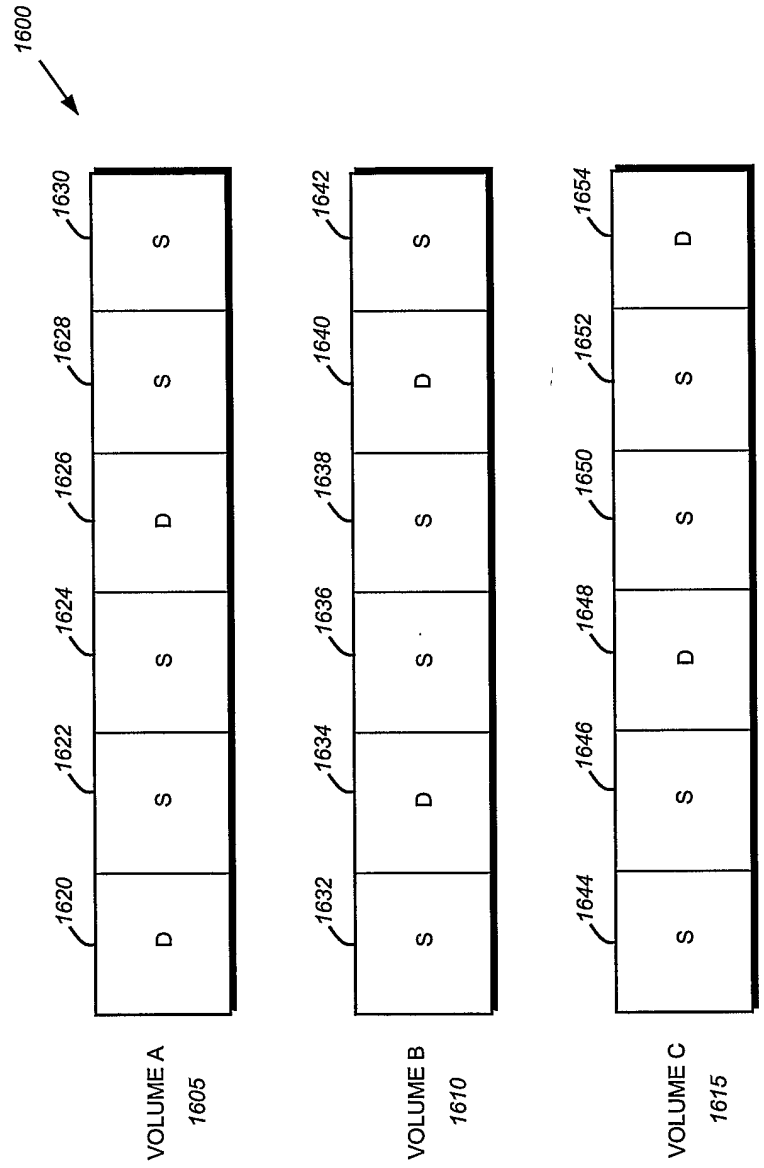

1600
1620
1622
1624
1626
1628
1630
D
S
S
D
S
S
VOLUME A
1605
1632
1634
1636
1638
1640
1642
S
D
S
S
D
S
VOLUME B
1610
1644
1646
1648
1650
1652
1654
S
S
D
S
S
D
VOLUME C
1615

FIG. 16

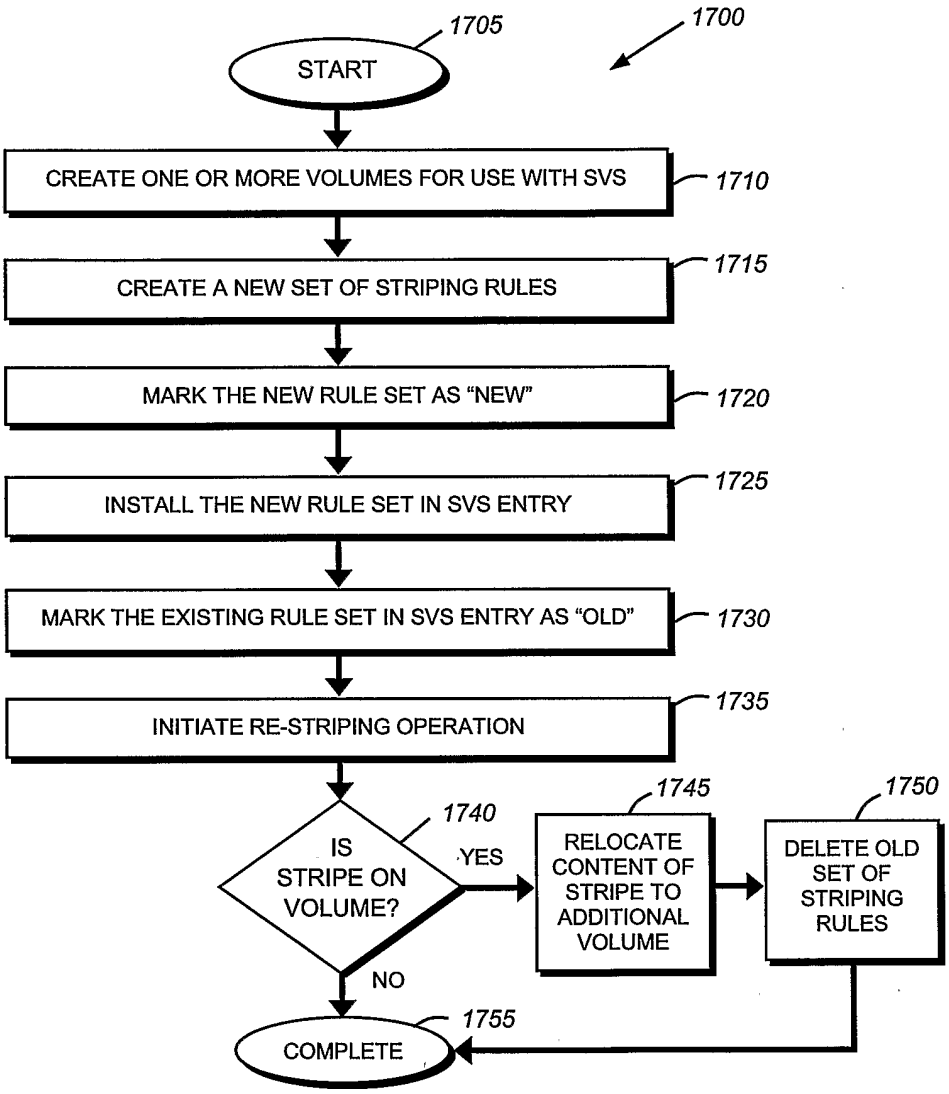

1700
1705
START
CREATE ONE OR MORE VOLUMES FOR USE WITH SVS
1710
CREATE A NEW SET OF STRIPING RULES
1715
MARK THE NEW RULE SET AS "NEW"
1720
INSTALL THE NEW RULE SET IN SVS ENTRY
1725
MARK THE EXISTING RULE SET IN SVS ENTRY AS "OLD"
1730
INITIATE RE-STRIPING OPERATION
1735
1740
IS STRIPE ON VOLUME?
YES
NO
1745
RELOCATE CONTENT OF STRIPE TO ADDITIONAL VOLUME
1750
DELETE OLD SET OF STRIPING RULES
1755
COMPLETE

FIG. 17

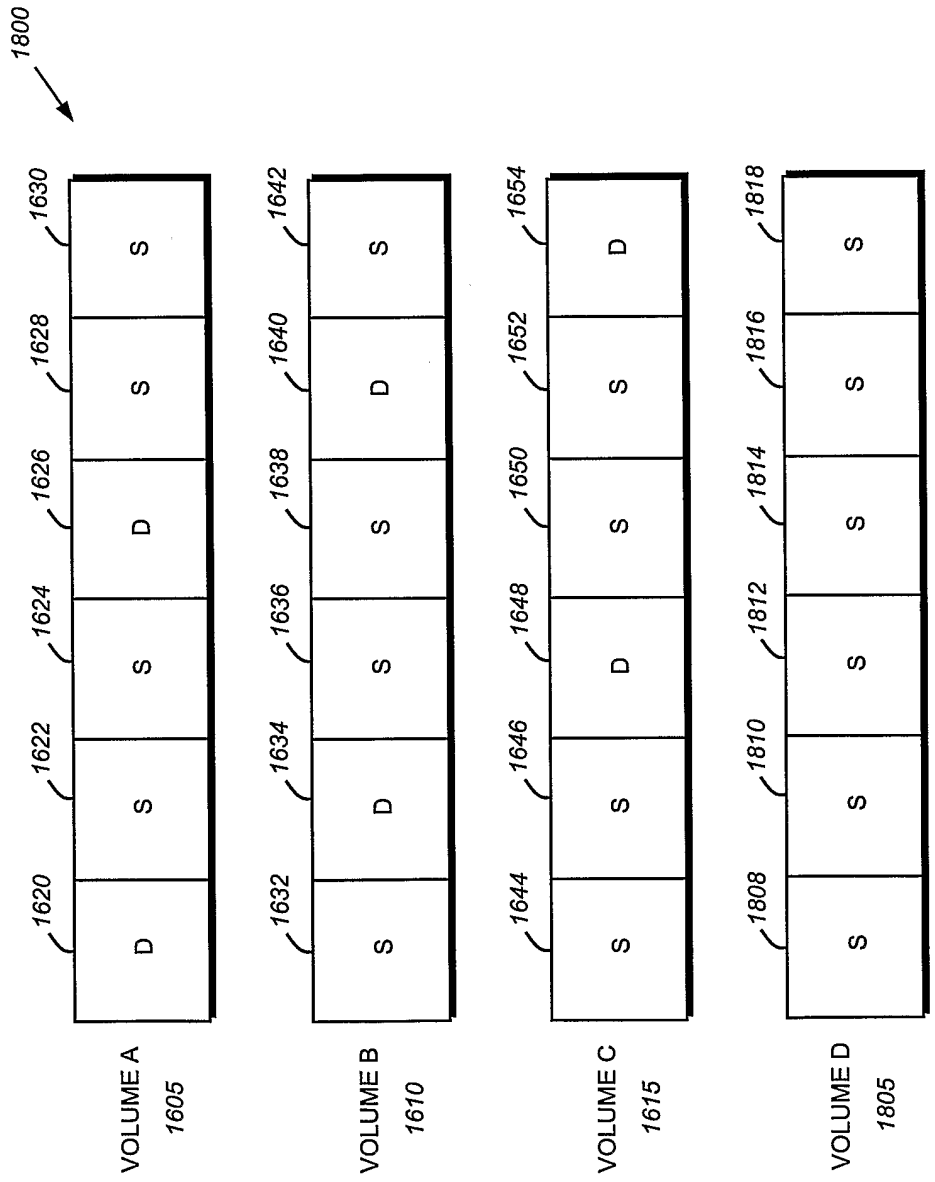

1800
1620
1622
1624
1626
1628
1630
D
S
S
D
S
S
VOLUME A
1605
1632
1634
1636
1638
1640
1642
S
D
S
S
D
S
VOLUME B
1610
1644
1646
1648
1650
1652
1654
S
S
D
S
S
D
VOLUME C
1615
1808
1810
1812
1814
1816
1818
S
S
S
S
S
S
VOLUME D
1805

FIG. 18

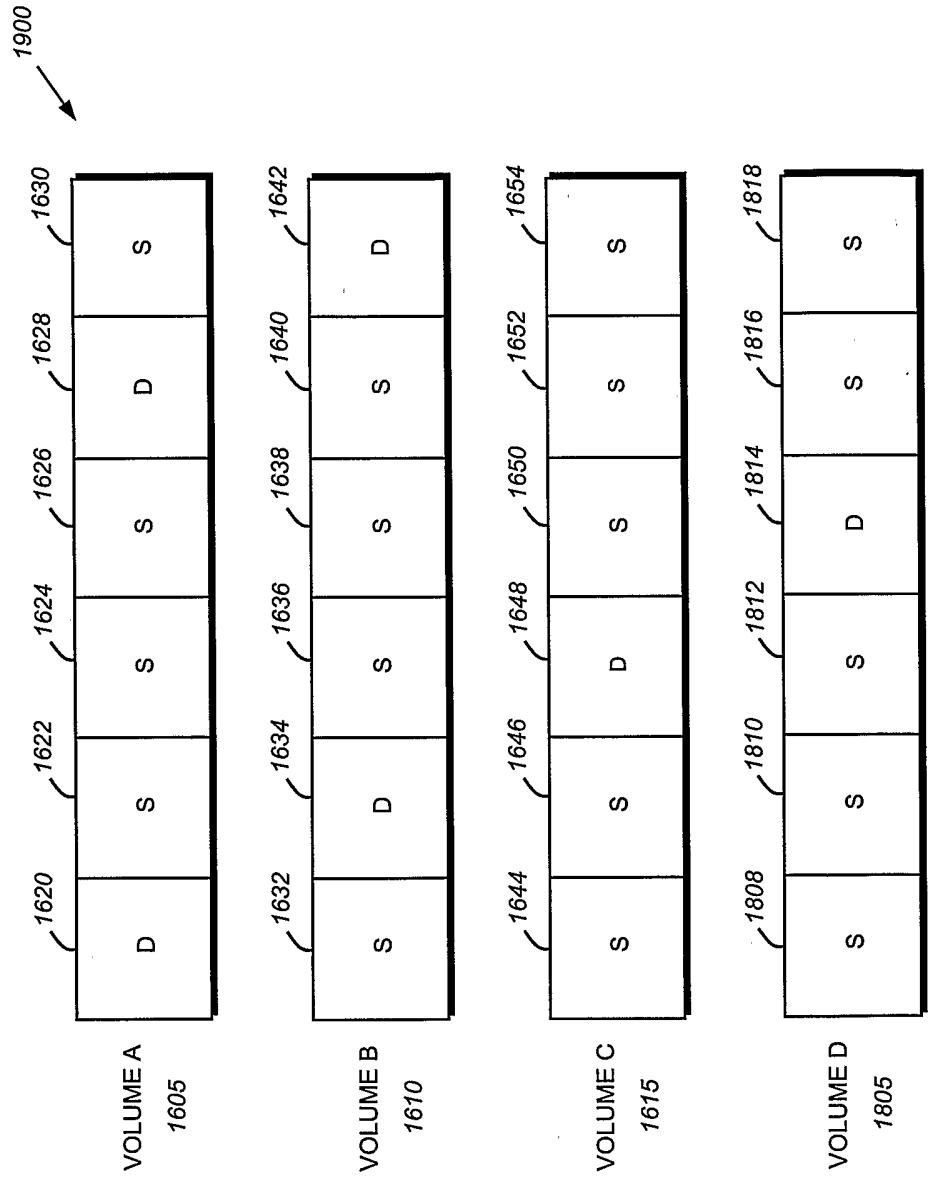

1900
1620
1622
1624
1626
1628
1630
D
S
S
S
D
S
VOLUME A
1605
1632
1634
1636
1638
1640
1642
S
D
S
S
S
D
VOLUME B
1610
1644
1646
1648
1650
1652
1654
S
S
D
S
S
S
VOLUME C
1615
1808
1810
1812
1814
1816
1818
S
S
S
D
S
S
VOLUME D
1805

FIG. 19

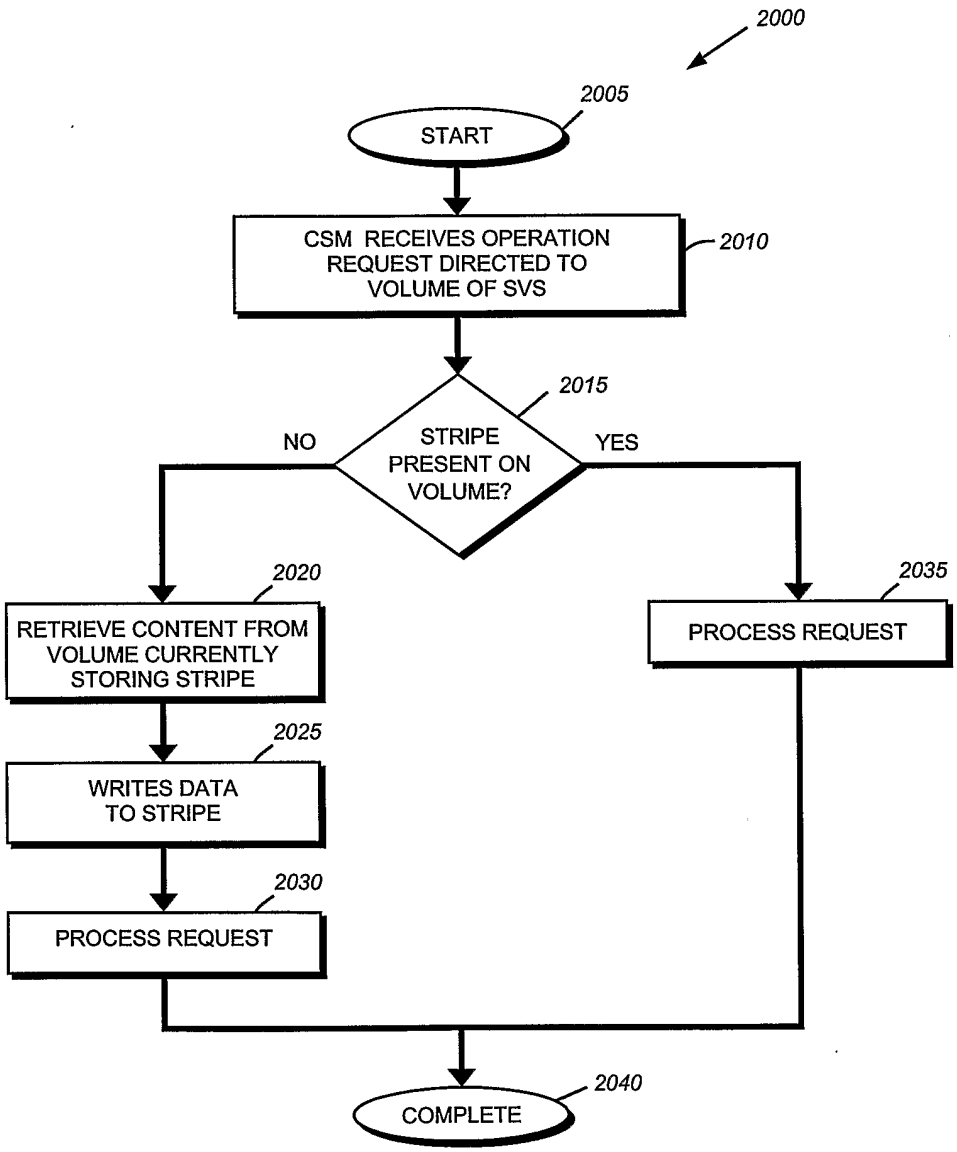

2000
2005
START
CSM RECEIVES OPERATION REQUEST DIRECTED TO VOLUME OF SVS
2010
2015
STRIPE PRESENT ON VOLUME?
NO
YES
2020
RETRIEVE CONTENT FROM VOLUME CURRENTLY STORING STRIPE
2025
WRITES DATA TO STRIPE
2030
PROCESS REQUEST
2035
PROCESS REQUEST
2040
COMPLETE

FIG. 20

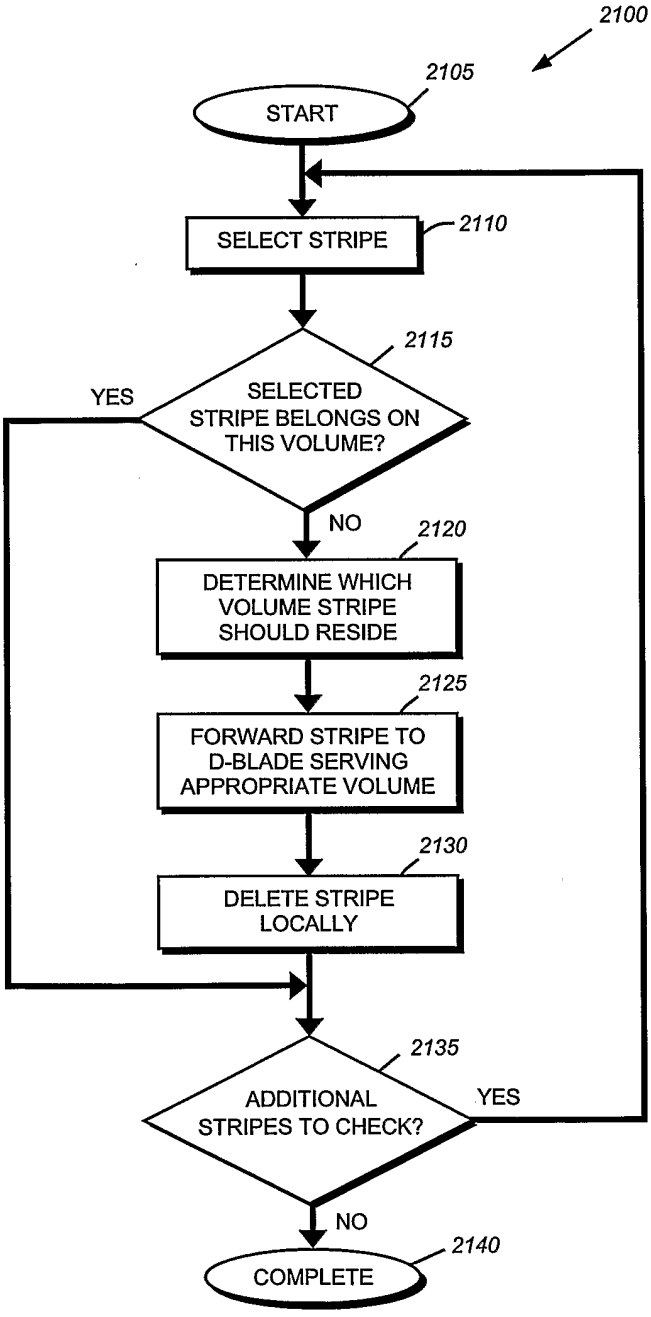

2100
2105
START
SELECT STRIPE
2110
2115
SELECTED STRIPE BELONGS ON THIS VOLUME?
YES
NO
2120
DETERMINE WHICH VOLUME STRIPE SHOULD RESIDE
2125
FORWARD STRIPE TO D-BLADE SERVING APPROPRIATE VOLUME
2130
DELETE STRIPE LOCALLY
2135
ADDITIONAL STRIPES TO CHECK?
YES
NO
2140
COMPLETE

FIG. 21

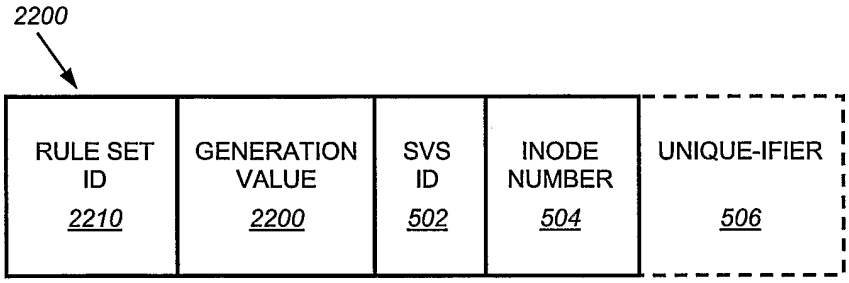

2200
RULE SET ID
2210
GENERATION VALUE
2200
SVS ID
502
INODE NUMBER
504
UNIQUE-IFIER
506

FIG. 22

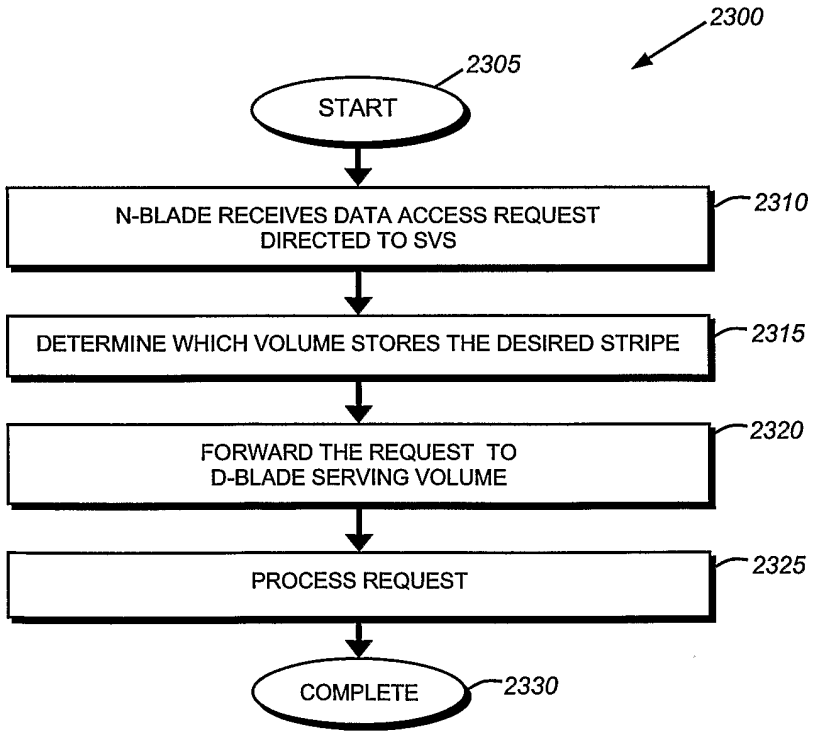

2300
2305
START
N-BLADE RECEIVES DATA ACCESS REQUEST DIRECTED TO SVS
2310
DETERMINE WHICH VOLUME STORES THE DESIRED STRIPE
2315
FORWARD THE REQUEST TO D-BLADE SERVING VOLUME
2320
PROCESS REQUEST
2325
COMPLETE
2330

FIG. 23

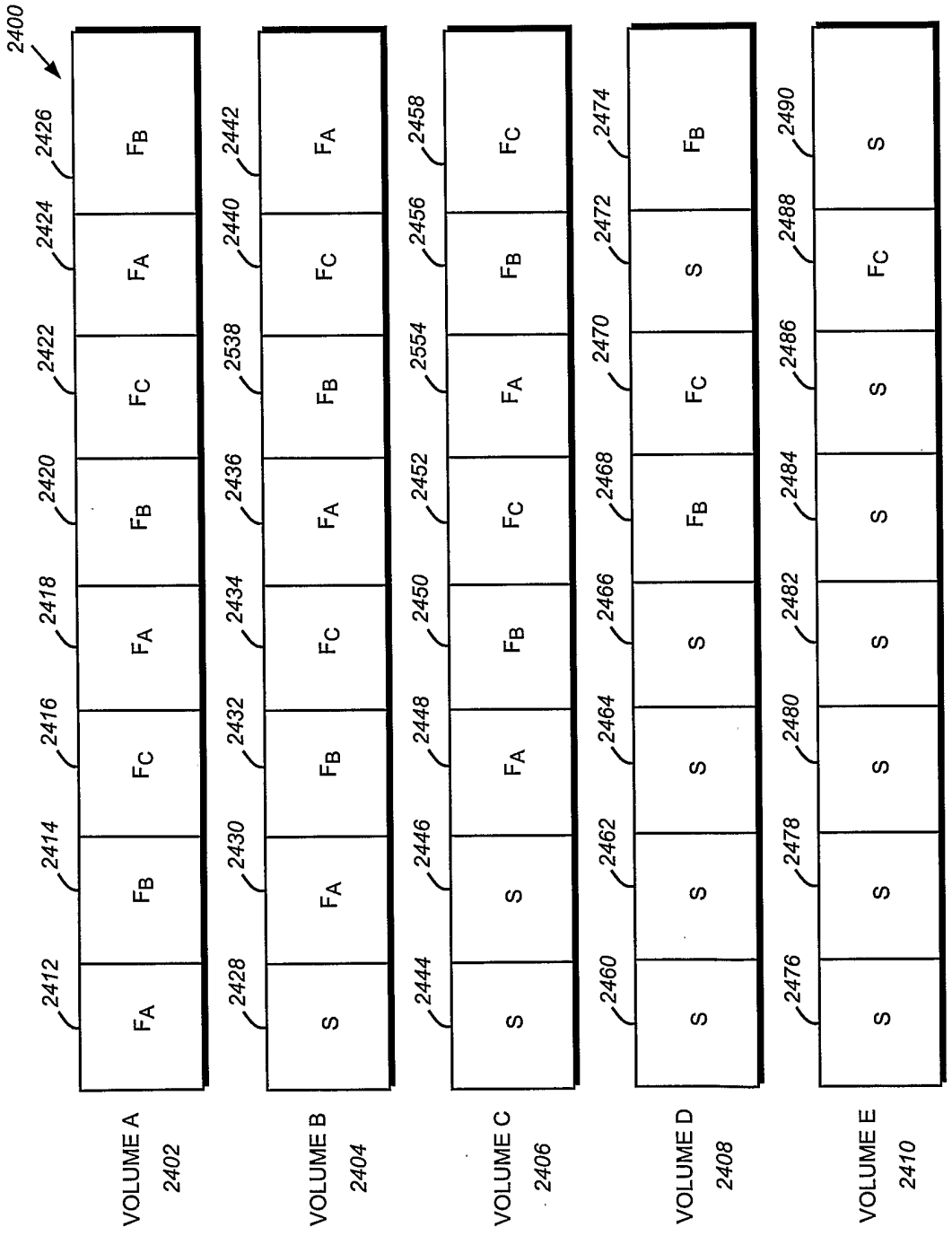

2400
VOLUME A 2402
VOLUME B 2404
VOLUME C 2406
VOLUME D 2408
VOLUME E 2410
2412 FA
2414 FB
2416 FC
2418 FA
2420 FB
2422 FC
2424 FA
2426 FB
2428 S
2430 FA
2432 FB
2434 FC
2436 FA
2538 FB
2440 FC
2442 FA
2444 S
2446 S
2448 FA
2450 FB
2452 FC
2554 FA
2456 FB
2458 FC
2460 S
2462 S
2464 S
2466 S
2468 FB
2470 FC
2472 S
2474 FB
2476 S
2478 S
2480 S
2482 S
2484 S
2486 S
2488 FC
2490 S

FIG. 24