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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
24 February 2005 (24.02.2005)

PCT

(10) International Publication Number
WO 2005/017737 A2

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

(21) International Application Number: PCT/US2004/026499

(22) International Filing Date: 13 August 2004 (13.08.2004)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/495,204	14 August 2003 (14.08.2003)	US
Not furnished	13 August 2004 (13.08.2004)	US

(71) Applicant *(for all designated States except US)*: **COMPELLENT TECHNOLOGIES** [US/US]; 12982 Valley View Road, Eden Prairie, MN 55344 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **SORAN, Philip, E.** [US/US]; 9501 Amesbury Lane, Eden Prairie, MN 55347 (US). **GUIDER, John, P.** [US/US]; 7 Catbird Lane, North Oaks, MN 55127 (US). **ASZMANN, Lawrence, E.** [US/US]; 5445 Shore Trail NE, Prior Lake, MN 55372 (US). **KLEMM, Michael, J.** [US/US]; 2301 Rivendell Lane, Minnetonka, MN 55305 (US).

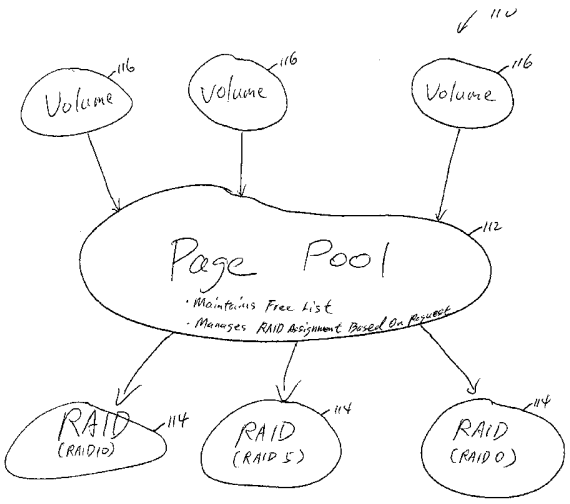
(74) Agents: **MIN, (Amy) Xu, S.** et al.; Dorsey & Whitney LLP, Intellectual Property Department, Suite 1500, 50 South Sixth Street, Minneapolis, MN 55402-1498 (US).

(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, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IT, LU, 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).

[Continued on next page]

(54) Title: VIRTUAL DISK DRIVE SYSTEM AND METHOD



(57) Abstract: A disk drive system and method capable of dynamically allocating data is provided. The disk drive system may include a RAID subsystem having a pool of storage, for example a page pool of storage that maintains a free list of RAIDs, or a matrix of disk storage blocks that maintain a null list of RAIDs, and a disk manager having at least one disk storage system controller. The RAID subsystem and disk manager dynamically allocate data across the pool of storage and a plurality of disk drives based on RAID-to-disk mapping. The RAID subsystem and disk manager determine whether additional disk drives are required, and a notification is sent if the additional disk drives are required. Dynamic data allocation and data progression allow a user to acquire a disk drive later in time when it is needed. Dynamic data allocation also allows efficient data storage of snapshots/point-in-time copies of virtual volume pool of storage, instant data replay and data instant fusion for data backup, recovery etc., remote data storage, and data progression, etc.

WO 2005/017737 A2

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.

VIRTUAL DISK DRIVE SYSTEM AND METHOD

FIELD OF THE INVENTION

The present invention generally relates to a disk drive system and method, and more particularly to a disk drive system having capabilities such as dynamic data allocation and disk drive virtualization, etc.

BACKGROUND OF THE INVENTION

The existing disk drive systems have been designed in such a way that a virtual volume data storage space is statically associated with physical disks with specific size and location for storing data. These disk drive systems need to know and monitor/control the exact location and size of the virtual volume of data storage space in order to store data. In addition, the systems often need bigger data storage space whereby more RAID devices are added. However, often times these additional RAID devices are expensive and not required until extra data storage space is actually needed.

Figure 14A illustrates a prior existing disk drive system having a virtual volume data storage space associated with physical disks with specific size and location for storing, reading/writing, and/or recovering data. The disk drive system statically allocates data based on the specific location and size of the virtual volume of data storage space. As a result, emptied data storage space is not used, and extra and sometimes expensive data storage devices, e.g. RAID devices, are acquired in advance for storing, reading/writing, and/or recovering

data in the system. These extra data storage space may not be needed and/or used until later in time.

Therefore, there is a need for an improved disk drive system and method. There is a further need for an efficient, dynamic data allocation and disk drive space and time management system and method.

SUMMARY OF THE INVENTION

The present invention provides an improved disk drive system and method capable of dynamically allocating data. The disk drive system may include a RAID subsystem having a matrix of disk storage blocks and a disk manager having at least one disk storage system controller. The RAID subsystem and disk manager dynamically allocate data across the matrix of disk storage blocks and a plurality of disk drives based on RAID-to-disk mapping. The RAID subsystem and disk manager determine whether additional disk drives are required, and a notification is sent if the additional disk drives are required. Dynamic data allocation allows a user to acquire a disk drive later in time when it is needed. Dynamic data allocation also allows efficient data storage of snapshots/point-in-time copies of virtual volume matrix or pool of disk storage blocks, instant data replay and data instant fusion for data backup, recovery etc., remote data storage, and data progression, etc. Data progression also allows deferral of a cheaper disk drive since it is purchased later in time.

In one embodiment, a matrix or pool of virtual volumes or disk storage blocks is provided to associate with physical disks. The matrix or pool of virtual

volumes or disk storage blocks is monitored/controlled dynamically by the plurality of disk storage system controllers. In one embodiment, the size of each virtual volume can be default or predefined by a user, and the location of each virtual volume is default as null. The virtual volume is null until data is allocated. The data can be allocated in any grid of the matrix or pool (e.g. a "dot" in the grid once data is allocated in the grid). Once the data is deleted, the virtual volume is again available as indicated to be "null". Thus, extra data storage space and sometimes expensive data storage devices, e.g. RAID devices, can be acquired later in time on a need basis.

In one embodiment, a disk manager may manage a plurality of disk storage system controllers, and a plurality of redundant disk storage system controllers can be implemented to cover the failure of an operated disk storage system controller.

In one embodiment, a RAID subsystem includes a combination of at least one of RAID types, such as RAID-0, RAID-1, RAID-5, and RAID-10. It will be appreciated that other RAID types can be used in alternative RAID subsystems, such as RAID-3, RAID-4, RAID-6, and RAID-7, etc.

The present invention also provides a dynamic data allocation method which includes the steps of: providing a default size of a logical block or disk storage block such that disk space of a RAID subsystem forms a matrix of disk storage blocks; writing data and allocating the data in the matrix of the disk storage blocks; determining occupancy rate of the disk space of the RAID subsystem based on historical occupancy rate of the disk space of the RAID

subsystem; determining whether additional disk drives are required; and sending a notification to the RAID subsystem if the additional disk drives are required. In one embodiment, the notification is sent via an email.

One of the advantages of the disk drive system of the present invention is that the RAID subsystem is capable of employing RAID techniques across a virtual number of disks. The remaining storage space is freely available. Through monitoring storage space and determining occupancy rate of the storage space of the RAID subsystem, a user does not have to acquire a large sum of drives that are expensive but has no use at the time of purchase. Thus, adding drives when they are actually needed to satisfy the increasing demand of the storage space would significantly reduce the overall cost of the disk drives. Meanwhile, the efficiency of the use of the drives is substantially improved.

Another advantage of the present invention is that the disk storage system controller is universal to any computer file system, not just to a specific computer file system.

The present invention also provides a method of data instant replay. In one embodiment, the data instant replay method includes the steps of: providing a default size of a logical block or disk storage block such that disk space of a RAID subsystem forms a page pool of storage or a matrix of disk storage blocks; automatically generating a snapshot of volumes of the page pool of storage or a snapshot of the matrix of disk storage blocks at predetermined time intervals; and storing an address index of the snapshot or delta in the page pool of storage or the

matrix of the disk storage blocks such that the snapshot or delta of the matrix of the disk storage blocks can be instantly located via the stored address index.

The data instant replay method automatically generates snapshots of the RAID subsystem at user defined time intervals, user configured dynamic time stamps, for example, every few minutes or hours, etc., or time directed by the server. In case of a system failure or virus attack, these time-stamped virtual snapshots allow data instant replay and data instant recovery in a matter of a few minutes or hours, etc. The technique is also referred to as instant replay fusion, i.e. the data shortly before the crash or attack is fused in time, and the snapshots stored before the crash or attack can be instantly used for future operation.

In one embodiment, the snapshots can be stored at a local RAID subsystem or at a remote RAID subsystem so that if a major system crash occurs due to, for example a terrorist attack, the integrity of the data is not affected, and the data can be instantly recovered.

Another advantage of the data instant replay method is that the snapshots can be used for testing while the system remains its operation. Live data can be used for real-time testing.

The present invention also provides a system of data instant replay including a RAID subsystem and a disk manager having at least one disk storage system controller. In one embodiment, the RAID subsystem and disk manager dynamically allocate data across disk space of a plurality of drives based on RAID-to-disk mapping, wherein the disk space of the RAID subsystem forms a matrix of disk storage blocks. The disk storage system controller automatically

generates a snapshot of the matrix of disk storage blocks at predetermined time intervals and stores an address index of the snapshot or delta in the matrix of the disk storage blocks such that the snapshot or delta of the matrix of the disk storage blocks can be instantly located via the stored address index.

In one embodiment, the disk storage system controller monitors frequency of data use from the snapshots of the matrix of the disk storage blocks and applies an aging rule such that the less frequently used or accessed data is moved to the less expensive RAID subsystem. Similarly, when the data in the less expensive RAID subsystem starts to be used more frequently, the controller moves the data to the more expensive RAID subsystem. Accordingly, a user is able to choose a desired RAID subsystem portfolio to meet its own storage needs. Therefore, the cost of the disk drive system can be significantly reduced and dynamically controlled by a user.

These and other features and advantages of the present invention will become apparent to those skilled in the art from the following detailed description, wherein it is shown and described illustrative embodiments of the invention, including best modes contemplated for carrying out the invention. As it will be realized, the invention is capable of modifications in various obvious aspects, all without departing from the spirit and scope of the present invention. Accordingly, the drawings and detailed description are to be regarded as illustrative in nature and not restrictive.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 illustrates one embodiment of a disk drive system in a computer environment in accordance with the principles of the present invention.

Figure 2 illustrates one embodiment of a dynamic data allocation having a page pool of storage for a RAID subsystem of a disk drive in accordance with the principles of the present invention.

Figure 2A illustrates a conventional data allocation in a RAID subsystem of a disk drive system.

Figure 2B illustrates a data allocation in a RAID subsystem of a disk drive system in accordance with the principles of the present invention.

Figure 2C illustrates a dynamic data allocation method in accordance with the principles of the present invention.

Figures 3A and 3B are schematic views of a snapshot of a disk storage block of a RAID subsystem at a plurality of time-intervals in accordance with the principles of the present invention.

Figure 3C illustrates a data instant replay method in accordance with the principles of the present invention.

Figure 4 is a schematic view of a data instant fusion function by using snapshots of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention.

Figure 5 is a schematic view of a local-remote data replication and instant replay function by using snapshots of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention.

Figure 6 illustrates a schematic view of a snapshot using the same RAID interface to perform I/O and concatenating multiple RAID devices into a volume in accordance with the principles of the present invention.

Figure 7 illustrates one embodiment of a snapshot structure in accordance with the principles of the present invention.

Figure 8 illustrates one embodiment of a PITC life cycle in accordance with the principles of the present invention.

Figure 9 illustrates one embodiment of a PITC table structure having a multi-level index in accordance with the principles of the present invention.

Figure 10 illustrates one embodiment of recovery of a PITC table in accordance with the principles of the present invention.

Figure 11 illustrates one embodiment of a write process having an owned page sequence and a non-owned page sequence in accordance with the principles of the present invention.

Figure 12 illustrates an exemplary snapshot operation in accordance with the principles of the present invention.

Figure 13A illustrates a prior existing disk drive system having a virtual volume data storage space associated with physical disks with specific size and location for statically allocating data.

Figure 13B illustrates a volume logical block mapping in the prior existing disk drive system of Figure 13A.

Figure 14A illustrates one embodiment of a disk drive system having a virtual volume matrix of disk storage blocks for dynamically allocating data in the system in accordance with the principles of the present invention.

Figure 14B illustrates one embodiment of dynamic data allocation in the virtual volume matrix of disk storage blocks as shown in Figure 14A.

Figure 14C illustrates a schematic view of a volume-RAID page remapping of one embodiment of the virtual volume page pool of storage in accordance with the principles of the present invention.

Figure 15 illustrates an example of three disk drives mapped to a plurality of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention.

Figure 16 illustrates an example of remapping of the disk drive storage blocks after adding a disk drive to three disk drives as shown in Figure 15.

Figure 17 illustrates one embodiment of accessible data pages in a data progression operation in accordance with the principles of the present invention.

Figure 18 illustrates a flow chart of one embodiment of a data progression process in accordance with the principles of the present invention.

Figure 19 illustrates one embodiment of compressed page layout in accordance with the principles of the present invention.

Figure 20 illustrates one embodiment of data progression in a high level disk drive system in accordance with the principles of the present invention.

Figure 21 illustrates one embodiment of external data flow in the subsystem in accordance with the principles of the present invention.

Figure 22 illustrates one embodiment of internal data flow in the subsystem.

Figure 23 illustrates one embodiment of each subsystem independently maintaining coherency.

Figure 24 illustrates one embodiment of a mixed RAID waterfall data progression in accordance with the principles of the present invention.

Figure 25 illustrates one embodiment of multiple free lists of a page pool of storage in accordance with the principles of the present invention.

Figure 26 illustrates one embodiment of a database example in accordance with the principles of the present invention.

Figure 27 illustrates one embodiment of a MRI image example in accordance with the principles of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention provides an improved disk drive system and method capable of dynamically allocating data. The disk drive system may include a RAID subsystem having a page pool of storage that maintains a free list of RAIDs or alternatively, a matrix of disk storage blocks, and a disk manager having at least one disk storage system controller. The RAID subsystem and disk manager dynamically allocate data across the page pool of storage or the matrix of disk storage blocks and a plurality of disk drives based on RAID-to-disk mapping. The RAID subsystem and disk manager determine whether additional disk drives are required, and a notification is sent if the additional disk drives are required. Dynamic data allocation allows a user to acquire a disk drive later in

time when it is needed. Dynamic data allocation also allows efficient data storage of snapshots/point-in-time copies of virtual volume matrix or pool of disk storage blocks, instant data replay and data instant fusion for data backup, recovery etc., remote data storage, and data progression, etc. Data progression also allows deferral of a cheaper disk drive since it is purchased later in time.

Figure 1 illustrates one embodiment of a disk drive system 100 in a computer environment 102 in accordance with the principles of the present invention. As shown in Figure 1, the disk drive system 100 includes a RAID subsystem 104 and a disk manager 106 having at least one disk storage system controller (Figure 16). The RAID subsystem 104 and disk manager 106 dynamically allocate data across disk space of a plurality of disk drives 108 based on RAID-to-disk mapping. In addition, the RAID subsystem 104 and disk manager 106 are capable of determining whether additional disk drives are required based on the data allocation across disk space. If the additional disk drives are required, a notification is sent to a user so that additional disk space may be added if desired.

The disk drive system 100 having a dynamic data allocation (or referred to "disk drive virtualization") in accordance with the principles of the present invention is illustrated in Figure 2 in one embodiment and Figures 14A and 14B in another embodiment. As shown in Figure 2, a disk storage system 110 includes a page pool of storage 112, i.e. a pool of data storage including a list of data storage space that is free to store data. The page pool 112 maintains a free list of RAID devices 114 and manages read/write assignments based on user's

requests. User's requested data storage volumes 116 are sent to the page pool 112 to get storage space. Each volume can request same or different classes of storage devices with same or different RAID levels, e.g. RAID 10, RAID 5, RAID 0, etc.

Another embodiment of dynamic data allocation of the present invention is shown in Figures 14A and 14B, where a disk storage system 1400 having a plurality of disk storage system controllers 1402 and a matrix of disk storage blocks 1404 controlled by the plurality of disk storage system controllers 1402 dynamically allocates data in the system in accordance with the principles of the present invention. The matrix of virtual volumes or blocks 1404 is provided to associate with physical disks. The matrix of virtual volumes or blocks 1404 is monitored/controlled dynamically by the plurality of disk storage system controllers 1402. In one embodiment, the size of each virtual volume 1404 can be predefined, for example 2 Megabytes, and the location of each virtual volume 1404 is default as null. Each of the virtual volumes 1404 is null until data is allocated. The data can be allocated in any grid of the matrix or pool (e.g. a "dot" in the grid once data is allocated in the grid). Once the data is deleted, the virtual volume 1404 is again available as indicated to be "null". Thus, extra and sometimes expensive data storage devices, e.g. RAID devices, can be acquired later in time on a need basis.

Accordingly, the RAID subsystem is capable of employing RAID techniques across a virtual number of disks. The remaining storage space is freely available. Through monitoring storage space and determining occupancy

rate of the storage space of the RAID subsystem, a user does not have to acquire a large sum of drives that are expensive but has no use at the time of purchase. Thus, adding drives when they are actually needed to satisfy the increasing demand of the storage space would significantly reduce the overall cost of the disk drives. Meanwhile, the efficiency of the use of the drives is substantially improved.

Also, dynamic data allocation of the disk drive system of the present invention allows efficient data storage of snapshots/point-in-time copies of virtual volume page pool of storage or virtual volume matrix of disk storage blocks, instant data replay and data instant fusion for data recovery and remote data storage, and data progression.

The above features and advantages resulted from a dynamic data allocation system and method and the implementation thereof in the disk drive system 100 are discussed below in details:

Dynamic Data Allocation

Figure 2A illustrates a conventional data allocation in a RAID subsystem of a disk drive system, whereby emptied data storage space is captive and not capable of being allocated for data storage.

Figure 2B illustrates a data allocation in a RAID subsystem of a disk drive system in accordance with the principles of the present invention, whereby emptied data storage that is available for data storage is mixed together to form a page pool, e.g. a single page pool in one embodiment of the present invention.

Figure 2C illustrates a dynamic data allocation method 200 in accordance with the principles of the present invention. The dynamic data allocation method 200 includes a step 202 of defining a default size of a logical block or disk storage block such that disk space of a RAID subsystem forms a matrix of disk storage blocks; and a step 204 of writing data and allocating the data in a disk storage block of the matrix where the disk storage block indicates "null". The method further includes a step 206 of determining occupancy rate of the disk space of the RAID subsystem based on historical occupancy rate of the disk space of the RAID subsystem; and a step 208 of determining whether additional disk drives are required and if so, sending a notification to the RAID subsystem. In one embodiment, the notification is sent via an email. Further, the size of the disk storage block can be set as a default and changeable by a user.

In one embodiment, dynamic data allocation, sometimes referred to as "virtualization" or "disk space virtualization", efficiently handles a large number of read and write requests per second. The architecture may require the interrupt handlers to call a cache subsystem directly. Dynamic data allocation may not optimize requests as it does not queue them, but it may have a large number of pending requests at a time.

Dynamic data allocation may also maintain data integrity and protect the contents of the data for any controller failure. To do so, dynamic data allocation writes state information to RAID device for reliable storage.

Dynamic data allocation may further maintain the order of read and write requests and complete read or write requests in the exact order that the requests

were received. Dynamic data allocation provides for maximum system availability and supports remote replication of data to a different geographical location.

In addition, dynamic data allocation provides recovery capabilities from data corruption. Through snapshot, a user may view the state of a disk in the past.

Dynamic data allocation manages RAID devices and provides a storage abstraction to create and expand large devices.

Dynamic data allocation presents a virtual disk device to the servers; the device is called a volume. To the server, the volume acts the same. It may return different information for serial number, but the volumes behave essentially like a disk drive. A volume provides a storage abstraction of multiple RAID devices to create a larger dynamic volume device. A volume includes multiple RAID devices, allowing for the efficient use of disk space.

Figure 21 illustrates a prior existing volume logical block mapping. Figure 14C shows a volume-RAID page remapping of one embodiment of the virtual volume page pool of storage in accordance with the principles of the present invention. Each volume is broken into a set of pages, e.g. 1, 2, 3, etc., and each RAID is broken into a set of pages. The volume page size and the RAID page size can be the same in one embodiment. Accordingly, one example of the volume-RAID page remapping of the present invention is that page #1 using a RAID-2 is mapped to RAID page #1.

Dynamic data allocation maintains data integrity of the volumes. Data is written to the volumes and confirmed to the server. Data integrity covers various controller configurations including stand alone and redundant through a controller failure. Controller failure includes power failure, power cycle, software exception, and hard reset. Dynamic data allocation generally does not handle disk drive failures which are covered by RAID.

Dynamic data allocation provides the highest levels of data abstraction for the controller. It accepts requests from the front end and ultimately uses RAID devices to write the data to disks.

Dynamic data allocation includes a number of internal subsystems:

- Cache – Smoothes read and write operations to a volume by providing rapid response time to the server, and bundling writes to data plug-in.
- Configuration – Contains the methods to create, delete, retrieve, and modify data allocation objects. Provides components to create a toolbox for higher level system applications.
- Data Plug-In – Distributes volume read and write requests to various subsystems depending on volume configuration.
- RAID Interface – Provides RAID device abstraction to create larger volumes to the user and other dynamic data allocation subsystems.
- Copy/Mirror/Swap – Replicates volume data to local and remote volumes. In one embodiment, it may only copy the blocks written by the server.
- Snapshot – Provides incremental volume recovery of data. It instantly creates View Volumes of past volume states.

- Proxy Volume – Implements request communication to a remote destination volume to support the Remote Replication.
- Billing – Ability to charge users for allocated storage, activity, performance, and recovery of data.

Dynamic data allocation also logs any errors and significant changes in configuration.

Figure 21 illustrates one embodiment of external data flow in the subsystem. External requests come from Front End. Requests include get volume information, read and write. All requests have the volume ID. Volume information is handled by the volume configuration subsystem. Read and write requests include the LBA. Write requests also include the data.

Depending on volume configuration, dynamic data allocation passes a request to a number of external layers. Remote replication passes requests to the front end, destined for a remote destination volume. The RAID Interface passes requests to RAID. Copy/mirror/swap passes requests back to dynamic data allocation to a destination volume.

Figure 22 illustrates one embodiment of internal data flow in the subsystem. The internal data flow starts with caching. Caching may place write requests into the cache or pass the requests directly to data plug-in. The cache supports direct DMA from front end HBA devices. Requests may be completed quickly and responses returned to the server. The data plug-in manager is the center of request flow below the cache. For each volume, it calls registered subsystem objects for each request.

Dynamic data allocation subsystems that affect data integrity may require support for controller coherency. As shown in Figure 23, each subsystem independently maintains coherency. Coherency updates avoid copying data blocks across the coherency link. Cache coherency may require copying data to the peer controller.

Disk Storage System Controller

Figure 14A illustrates a disk storage system 1400 having a plurality of disk storage system controllers 1402 and a matrix of disk storage blocks or virtual volumes 1404 controlled by the plurality of disk storage system controllers 1402 for dynamically allocating data in the system in accordance with the principles of the present invention. Figure 14B illustrates one embodiment of dynamic data allocation in the virtual volume matrix of disk storage blocks or virtual volumes 1404.

In one operation, the disk storage system 1400 automatically generates a snapshot of the matrix of disk storage blocks or virtual volumes 1404 at predetermined time intervals and stores an address index of the snapshot or delta in the matrix of the disk storage blocks or virtual volumes 1404 such that the snapshot or delta of the matrix of the disk storage blocks or virtual volumes 1404 can be instantly located via the stored address index.

Further in one operation, the disk storage system controller 1402 monitors frequency of data use from the snapshots of the matrix of the disk storage blocks 1404 and applies an aging rule such that the less frequently used or accessed data

is moved to the less expensive RAID subsystem. Similarly, when the data in the less expensive RAID subsystem starts to be used more frequently, the controller moves the data to the more expensive RAID subsystem. Accordingly, a user is able to choose a desired RAID subsystem portfolio to meet its own storage needs. Therefore, the cost of the disk drive system can be significantly reduced and dynamically controlled by a user.

RAID-to-Disk Mapping

A RAID subsystem and disk manager dynamically allocate data across disk space of a plurality of disk drives based on RAID-to-disk mapping. In one embodiment, the RAID subsystem and disk manager determine whether additional disk drives are required, and a notification is sent if the additional disk drive is required.

Figure 15 illustrates an example of three disk drives 108 (Figure 1) mapped to a plurality of disk storage blocks 1502-1512 in a RAID-5 subsystem 1500 in accordance with the principles of the present invention.

Figure 16 illustrates an example of remapping 1600 of the disk drive storage blocks after adding a disk drive 1602 to three disk drives 108 as shown in Figure 15.

Disk Manager

The disk manager 106, as shown in Figure 1, generally manages disks and disk arrays, including grouping/resource pooling, abstraction of disk attributes,

formatting, addition/subtraction of disks, and tracking of disk service times and error rates. The disk manager 106 does not distinguish the differences between various models of disks and presents a generic storage device for the RAID component. The disk manager 106 also provides grouping capabilities which facilitate the construction of RAID groups with specific characteristics such as 10,000 RPM disks, etc.

In one embodiment of the present invention, the disk manager 106 is at least three-fold: abstraction, configuration, and I/O optimization. The disk manager 106 presents "disks" to upper layers which could be, for example, locally or remotely attached physical disk drives, or remotely attached disk systems.

The common underlying characteristic is that any of these devices could be the target of I/O operations. The abstraction service provides a uniform data path interface for the upper layers, particularly the RAID subsystem, and provides a generic mechanism for the administrator to manage target devices.

The disk manager 106 of the present invention also provides disk grouping capabilities to simplify administration and configuration. Disks can be named, and placed into groups, which can also be named. Grouping is a powerful feature which simplifies tasks such as migrating volumes from one group of disks to another, dedicating a group of disks to a particular function, specifying a group of disks as spares, etc.

The disk manager also interfaces with devices, such as a SCSI device subsystem which is responsible for detecting the presence of external devices. The SCSI device subsystem is capable, at least for fiber channel/SCSI type devices, of determining a subset of devices which are block-type target devices. It is these devices which are managed and abstracted by the disk manager.

Further, the disk manager is responsible for responding to flow control from a SCSI device layer. The disk manager has queuing capabilities, which presents the opportunity to aggregate I/O requests as a method to optimize the throughput of the disk drive system.

Furthermore, the disk manager of the present invention manages a plurality of disk storage system controllers. Also, a plurality of redundant disk storage system controllers can be implemented to cover the failure of an operated disk storage system controller. The redundant disk storage system controllers are also managed by the disk manager.

Disk Manager's Relationship to the Other Subsystems

The disk manager interacts with several other subsystems. The RAID subsystem is the major client of the services provided by the disk manager for data path activities. The RAID subsystem uses the disk manager as the exclusive path to disks for I/O. The RAID system also listens for events from the disk manager to determine the presence and operational status of disks. The RAID subsystem also works with the disk manager to allocate extents for the construction of RAID devices. Management control listens for disk events to

learn the existence of disks and to learn of operational status changes. In one embodiment of the present invention, the RAID subsystem 104 may include a combination of at least one of RAID types, such as RAID-0, RAID-1, RAID-5, and RAID-10. It will be appreciated that other RAID types can be used in alternative RAID subsystems, such as RAID-3, RAID-4, RAID-6, and RAID-7, etc.

In one embodiment of the present invention, the disk manager utilizes the services of configuration access to store persistent configuration and present transient read-only information such as statistics to the presentations layers. The disk manager registers handlers with configuration access for access to these parameters.

The disk manager also utilizes the services of the SCSI device layer to learn of the existence and operational status of block devices, and has an I/O path to these block devices. The disk manager queries the SCSI device subsystem about devices as a supporting method to uniquely identify disks.

Data Instant Replay and Data Instant Fusion

The present invention also provides a method of data instant replay and data instant fusion. Figures 3A and 3B illustrate schematic views of a snapshot of a disk storage block of a RAID subsystem at a plurality of time-intervals in accordance with the principles of the present invention. Figure 3C illustrates a data instant replay method 300 which includes a step 302 of defining a default size of a logical block or disk storage block such that disk space of a RAID

subsystem forms a page pool of storage or a matrix of disk storage blocks; a step 304 of automatically generating a snapshot of volumes of the page pool or a snapshot of the matrix of disk storage blocks at predetermined time intervals; and storing an address index of the snapshot or delta in the page pool of storage or the matrix of the disk storage blocks such that the snapshot or delta of the matrix of the disk storage blocks can be instantly located via the stored address index.

As shown in Figure 3B, at each predetermined time interval, e.g. 5 minutes, such as T1 (12:00 PM), T2 (12:05 PM), T3 (12:10 PM), and T4 (12:15 PM), a snapshot of the page pool of storage or the matrix of disk storage blocks are automatically generated. The address indexes of the snapshots or delta in the page pool of storage or the matrix of the disk storage blocks are stored in the page pool of storage or the matrix of the disk storage blocks such that the snapshot or delta of the page pool of storage or the matrix of the disk storage blocks can be instantly located via the stored address index.

Accordingly, the data instant replay method automatically generates snapshots of the RAID subsystem at a user defined time intervals, user configured dynamic time stamps, for example, every few minutes or hours, etc., or time directed by the server. In case of a system failure or virus attack, these time-stamped virtual snapshots allow data instant replay and data instant recovery in a matter of a few minutes or hours, etc. The technique is also referred to as instant replay fusion, i.e. the data shortly before the crash or attack is fused in time, and the snapshots stored before the crash or attack can be instantly used for future operation.

Figure 4 further illustrates a schematic view of a data instant fusion function 400 by using multiple snapshots of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention. At T3, a parallel chain T3'-T5' of snapshots are generated, whereby data that are fused and/or recovered by the fused data T3' can be used to replace the to-be-fused data at T4. Similarly, a plurality of parallel chains T3", T4''' of snapshots can be generated to replace the to-be-fused data at T4'-T5' and T4"-T5". In an alternative embodiment, the snapshots at T4, T4'-T5', T5" can still be stored in the page pool or the matrix.

The snapshots can be stored at a local RAID subsystem or at a remote RAID subsystem so that if a major system crash occurs due to, for example a terrorist attack, the integrity of the data is not affected, and the data can be instantly recovered. Figure 5 illustrates a schematic view of a local-remote data replication and instant replay function 500 by using snapshots of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention.

Remote replication performs the service of replicating volume data to a remote system. It attempts to keep the local and remote volumes as closely synchronized as possible. In one embodiment, the data of the remote volume may not mirror a perfect copy of the data of the local volume. Network connectivity and performance may cause the remote volume to be out of synchronization with a local volume.

Another feature of the data instant replay and data instant fusion method is that the snapshots can be used for testing while the system remains its operation. Live data can be used for real-time testing.

Snapshot and Point-in-Time Copies (PITC)

An example of data instant replay is to utilize snapshots of disk storage blocks of a RAID subsystem in accordance with the principles of the present invention. Snapshot records write operations to a volume so that a view may be created to see the contents of a volume in the past. Snapshot thus also supports data recovery by creating views to a previous Point-in-Time Copy (PITC) of a volume.

The core of a snapshot implements create, coalesce, management, and I/O operations of the snapshot. Snapshot monitors writes to a volume and creates Point-in-Time Copies (PITC) for access through view volumes. It adds a Logical Block Address (LBA) remapping layer to a data path within the virtualization layer. This is another layer of virtual LBA mapping within the I/O path. The PITC may not copy all volume information, and it may merely modify a table that the remapping uses.

Snapshot tracks changes to volume data and provides the ability to view the volume data from a previous point-in-time. Snapshot performs this function by maintaining a list of delta writes for each PITC.

Snapshot provides multiple methods for PITC profiles including: application initiated, and time initiated. Snapshot provides the ability for the

application to create PITC. The applications control the creation through the API on the server, which is delivered to the snapshot API. Also, snapshot provides the ability to create a time profile.

Snapshot may not implement a journaling system or recover all writes to a volume. Snapshot may only keep the last write to a single address within a PITC window. Snapshot allows a user to create PITC that covers a defined short period of time, such as minutes or hours, etc. To handle failures, snapshot writes all information to disk. Snapshot maintains volume data page pointers containing the delta writes. Since the tables provide the map to the volume data, and without it the volume data is inaccessible, the table information must handle controller failure cases.

View volume functions provide access to a PITC. View volume functions may attach to any PITC within the volume, except the active PITC. Attaching to a PITC is a relatively quick operation. Uses of view volume functions include testing, training, backup, and recovery. The view volume functions allow write operation and do not modify the underlying PITC it is based on.

In one embodiment, the snapshot is designed to optimize performance and ease use at the expense of disk space:

- Snapshot provides speedy response time for user requests. User requests include I/O, create a PITC, and create/delete a view volume. To achieve this snapshot uses more disk space to store table information than the minimum required. For I/O, snapshot summarizes the current state of a volume into a single table, so that all read and write requests may be

satisfied by a single table. Snapshot reduces the impact on normal I/O operations as much as possible. Second, for view volume operations snapshot uses the same table mechanism as the main volume data path.

- Snapshot minimizes the amount of data copied. To do this, snapshot maintains a table of pointers for each PITC. Snapshot copies and moves pointers, but it does not move the data on the volume.
- Snapshot manages the volume using fixed-size data-pages. Tracking individual sectors may require massive amounts of memory for a single reasonable sized volume. By using a data page larger than a sector certain pages may contain a percentage of information directly duplicated from another page.
- Snapshot uses the data space on the volume to store the data-page tables. The lookup tables are reproduced after a controller failure. The lookup tables allocate pages and sub-divide them.
- Snapshot handles controller failure by requiring that a volume using snapshot operate on a single controller in one embodiment. This embodiment requires no coherency. All changes to the volume are recorded on disk or to reliable cache for recovery by a replacement controller. Recovery from a controller failure requires that the snapshot information be read from disk in one embodiment.
- Snapshot uses the virtualization RAID interface to access the storage. Snapshot may use multiple RAID devices as a single data space.

- Snapshot supports 'n' PITC per volume and 'm' views per volume. The limitation on 'n' and 'm' is a function of the disk space and memory of the controller.

Volume and Volume Allocation/Layout

Snapshots add a LBA remapping layer to a volume. The remapping uses the I/O request LBA and the lookup table to convert the address to the data page. As shown in Figure 6, a presented volume using snapshot behaves the same as a volume without snapshot. It has a linear LBA space and handles I/O requests. Snapshot uses the RAID interface to perform I/O and includes multiple RAID devices into a volume. In one embodiment, the size of the RAID devices for a snapshot volume is not the size of the presented volume. The RAID devices allow snapshot to expand the space for data pages within the volume.

A new volume, with snapshot enabled at the inception, only needs to include space for the new data pages. Snapshot does not create a list of pages to place in the bottom level PITC. The bottom level PITC is empty in this case. At allocation, all PITC pages are on the free list. By creating a volume with snapshot enabled at the inception, it may allocate less physical space than the volume presents. Snapshot tracks the writes to the volume. In one embodiment of the present invention, the NULL volume is not copied and/or stored in the page pool or matrix, thereby increasing the efficiency of the use of the storage space.

In one embodiment, for both allocation schemes, PITC places a virtual NULL volume at the bottom of the list. Reads to the NULL volume return blocks of zero. The NULL volume handles the sectors not previously written by the server. Writes to the NULL volume can not occur. The volume uses a NULL volume for reads to unwritten sectors.

The number of free pages depends on the size of the volume, the number of PITC, and the expected rate of data change. The system determines the number of pages to allocate for a given volume. The number of data pages may expand over time. Expansion may support a more rapid change in data than expected, more PITC, or a larger volume. New pages are added to the free list. The addition of pages to the free list may occur automatically.

Snapshot uses data pages to manage the volume space. Each data page may include megabytes of data. Using the operating system tends to write a number of sectors in the same area of a volume. Memory requirements also dictate that snapshot uses pages to manage volumes. Maintaining a single 32-bit pointer for each sector of a one-terabyte volume may require eight gigabytes of RAM. Different volumes may have different page size.

Figure 7 illustrates one embodiment of a snapshot structure. Snapshot adds a number of objects to the volume structure. Additional objects include the PITC, a pointer to the active PITC, the data page free list, child view volumes, and PITC coalesce objects.

- Active PITC (AP) pointer is maintained by the volume. The AP handles the mapping of read and writes requests to the volume. The AP contains a summary of the current location of all the data within the volume.
- The data pages free list tracks the available pages on the volume.
- The optional child view volumes provide access to the volume PITC. The view volumes contain their own AP to record writes to the PITC, while not modifying the underlying data. A volume may support multiple child view volumes.
- Snapshot coalesce objects temporarily link two PITC for the purpose of removing the previous PITC. Coalescing of PITC involves moving the ownership of data pages and freeing of data pages.
- A PITC contains a table and data pages for the pages written while the PITC was active. The PITC contains a freeze time stamp at which point the PITC stopped accepting write requests. The PITC also contains a Time-to-Live value that determines at what time the PITC will coalesce.

Also, snapshot summarizes the data page pointers for the entire volume, at the time a PITC is taken to provide predictable read and write performance. Other solutions may require reads to examine multiple PITC to find the newest pointer. These solutions require table caching algorithm but has worst-case performance.

Snapshot summarizing in the present invention also reduces the worst-case memory usage of table. It may require that the entire table be loaded into memory, but it may require only a single table loaded.

The summary includes pages owned by the current PITC and may include pages from all previous PITC. To determine which pages the PITC may write, it tracks page ownership for each data page. It also tracks ownership for a coalesce process. To handle this, the data page pointer includes the page index.

Figure 8 illustrates one embodiment of a PITC life cycle. Each PITC goes through a number of following states before it is committed as read-only:

1. Create table – Upon creation, table is created.

2. Commit to disk – This generates the storage on the disk for the PITC. By writing the table at this point, it guarantees that the required space to store the table information is allocated before the PITC is taken. At the same time, the PITC object is also committed to the disk.

3. Accept I/O – It has become the active PITC (AP) – It now handles reads and writes requests for the volume. This is the only state that accepts writes requests to the table. The PITC generates an event that it is now active.

4. Commit the Table to Disk – The PITC is no longer the AP, and no longer accepts additional pages. A new AP has taken over. After this point, the table will not change unless it is removed during a coalesce operation. It is read-only. At this point, the PITC generates an event that it is frozen and committed. Any service may listen to the event.

5. Release table memory – Frees the memory that the table required. This step also clears the log to state that all changes are written to disk.

The top-level PITC for a volume or a view volume is called the active PITC (AP). The AP satisfies all read and write requests to the volume. The AP

is the only PITC for the volume that may accept write requests. The AP contains a summary of data page pointers for the entire volume.

The AP may be the destination, not the source, for a coalesce process. Being the destination, the AP increases the number of owned pages, but it does not change the view of the data.

For volume expansion, the AP immediately grows with the volume. The new pages point to the NULL volume. Non-AP PITC does not require modification for volume expansion.

Each PITC maintains a table to map an incoming LBA to a data page pointer to the underlying volume. The table includes pointers to data pages. The table needs to address more physical disk space than presented logical space. Figure 9 illustrates one embodiment of a table structure having a multi-level index. The structure decodes the volume LBA to a data-page pointer. Each level decodes increasing less significant bits of the address as shown in Figure 9. The structure of the table provides for fast lookup and the ability to expand the volume. For fast lookup, the multi-level index structure keeps the table shallow with multiple entries at each level. The index performs array lookups at each level. To support volume expansion, the multi-level index structure allows for the addition of another layer to support expansion. Volume expansion in this case is the expansion of the LBA count presented to the upper layer, and not the actual amount of storage space allocated for the volume.

The multi-level index contains a summary of the entire volume data page remapping. Each PITC contains a complete remapping list for the volume at the point-in-time it is committed.

The multi-level index structure uses different entry types for the levels of the table. The different entry types support the need to read the information from the disk, as well as store it in memory. The bottom level entries may only contain data page pointers. The top and middle level entries contain two arrays, one for the LBA of the next level table entry, and a memory pointer to the table.

As the presented volume size expands, the size of previous PITC tables does not need to increase, and the tables do not need to be modified. The information in the table may not change, since it is read only, and the expand process modifies the table by adding NULL page pointers to the end. Snapshot does not directly present the tables from previous PITC to the user.

An I/O operation asks the table to map an LBA to a data page pointer. The I/O then multiplies the data page pointer times the data page size to get the LBA of the underlying RAID. In one embodiment, data page size is a power of two.

The table provides an API to remap LBA, add page, and coalesce table.

Snapshot uses the data pages to store the PITC object and the LBA mapping tables. The tables directly access the RAID interface for I/O to its table entries. The table minimizes modification when reading and writing the table to the RAID device. Without modification, it becomes possible to read and write the table information directly into table entry structures. This reduces copies

needed for I/O. Snapshot may use a change log to prevent the creation of hot-spots on the disk. A hot-spot is a location that is used repeatedly to track updates to the volume. The change log records updates to the PITC table, and the free list for the volume. During recovery, snapshot uses the change log to re-create the in-memory AP and free list. Figure 10 illustrates one embodiment of recovery of a table, which demonstrates the relationship among the in-memory AP, the on-disk AP, and the change log. It also shows the same relationship for the free list. The in-memory AP table may be rebuilt from the on-disk AP table and the log. For any controller failure, the AP is rebuilt by reading the on-disk AP and applying the log changes to it. The change log uses different physical resources depending on system configuration. For multiple-controller systems, the change log relies on battery-backup cache memory for storage. Using cache memory allows snapshot to reduce the number of table writes to disk while maintaining data integrity. The change log replicates to a backup controller for recovery. For single-controller systems, the change log writes all information to the disks. This has the side-effect of creating a hot-spot on the disk at the log location. This allows a number of changes to be written to a single device block.

Periodically, snapshot writes the PITC table and free list to disk, creating a checkpoint in the log and clearing it. This period may vary depending on the number of updates to the PITC. The coalesce process does not use the change log.

Snapshot data page I/O may require requests fit within the data page boundaries. If snapshot encounters an I/O request that spans the page boundaries

it splits the request. It then passes the requests down to the request handlers. The write and read sections assume that an I/O fits within the page boundaries. The AP provides the LBA remapping to satisfy I/O requests.

The AP satisfies all write requests. Snapshot supports two different write sequences for owned and non-owned pages. The different sequence allow for the addition of pages to the table. Figure 11 illustrates one embodiment of a write process having an owned page sequence and a non-owned page sequence.

For the owned page sequence, the process includes the following:

1) Find the table mapping; and

2) Page Owned Write – Remap the LBA and write the data to the RAID interface.

A previously written page is the simple write request. Snapshot writes the data to the page, overwriting the current contents. Only data pages owned by the AP will be written. Pages owned by other PITC is read only.

For the non-owned page sequence, the process includes the following:

1) Find the table mapping;

2) Read previous Page – Perform a read to the data page such that the write request and the read data make up the complete page. This is the start of the copy on write process.

3) Combine the data – Put the data page read and the write request payloads into a single contiguous block.

4) Free List Allocate – Get a new data page pointer from the free list.

5) Write the combined data to the new data page.

6) Commit the new page information to the log.

7) Update the table – Change the LBA remapping in the table to reflect the new data page pointer. The data page is now owned by the PITC.

Adding a page may require blocking read and write requests until the page is added to the table. By writing the table updates to disk and keeping multiple cached copies of the log, snapshot achieves controller coherency.

With respect to read requests, the AP fulfills all read requests. Using the AP table the read request remaps the LBA to the LBA of the data page. It passes the remapped LBA to the RAID interface to satisfy the request. A volume may fulfill a read requests for a data page not previously written to the volume. These pages are marked with the NULL Address (All one's) in the PITC table. Requests to this address are satisfied by the NULL volume and return a constant data pattern. Pages owned by different PITC may satisfy a read request spanning page boundaries.

Snapshot uses a NULL volume to satisfy read requests to previously unwritten data pages. It returns all zeroes for each sector read. It does not have a RAID device or allocated space. It is anticipated that a block of all zeroes be kept in memory to satisfy the data requirements for a read to the NULL volume. All volumes share the NULL volume to satisfy read requests.

In one embodiment, a coalesce process removes a PITC and some of its owned pages from the volume. Removing the PITC creates more available space to track new differences. Coalescing compares two adjacent tables for

differences and keeps only the newer differences. Coalescing occurs periodically or manually according to user configuration.

The process may include two PITC, the source and destination. The rules in one embodiment for eligible objects are as follows:

1) The source must be the previous PITC to the Destination -- the source must be created before the destination.

2) A destination may not simultaneously be a source.

3) A source may not be referred to by multiple PITC. Multiple references occur when a view volume is created from a PITC.

4) The destination may support multiple references.

5) The AP may be a destination, but not a source.

The coalesce process writes all changes to disk and requires no coherency. If a controller fails, the volume recovers the PITC information from disk and resumes the coalesce process.

The process marks two PITC for coalescing and includes the following steps:

1) Source state set to coalesce source – the state is committed to disk for controller failure recovery. At his point source may no longer be accessed as its data pages may be invalid. The data pages may be returned to the free list, or ownership is transferred to destination.

2) Destination state set to coalesce destination – the state is committed to disk for controller failure recovery.

3) Load and compare tables – the process moves data page pointers. Freed data pages immediately are added to the free list.

4) Destination state set to normal – The process is complete.

5) Adjust the list – change the previous of the source next pointer to the destination. This effectively removes the source from the list.

6) Free the source – return any data pages used for control information to the free list.

The above process supports the combination of two PITC. It is appreciated to a person skilled in the art that coalesce can be designed to remove multiple PITC and create multiple sources in the single pass.

As shown in Figure 2, the page pool maintains a data page free list for use by all volumes associated with the page pool. The free list manager uses data pages from the page pool to commit the free list to permanent storage. Free list updates come from a number of sources: the write process allocates pages, the control page manager allocates pages, and the coalescing process returns pages.

The free list may maintain a trigger to automatically expand itself at a certain threshold. The trigger uses the page pool expansion method to add pages to the page pool. The automatic expansion could be a function of volume policy. More important data volume would be allowed to expand while less important volumes are forced to coalesce.

View volumes provide access to previous points-in-time and support normal volume I/O operations. A PITC tracks the difference between PITC, and the view volume allows the user to access the information contained within a

PITC. A view volume branches from a PITC. View volumes support recovery, test, backup operations, etc. View volume creation occurs nearly instantaneously as it requires no data copies. The view volume may require its own AP to support writes to the view volume.

A view taken from the current state of the volume the AP may be copied from the current volume AP. Using the AP, the view volume allows write operations to the view volume without modifying the underlying data. The OS may require a file system or file rebuild to use the data. The view volume allocates space from the parent volume for the AP and written data pages. The view volume has no associated RAID device information. Deleting the view volume frees the space back to the parent volume.

Figure 12 illustrates an exemplary snapshot operation showing the transitions for a volume using snapshot. Figure 12 depicts a volume with ten pages. Each state includes a Read Request Fulfillment list for the volume. Shaded blocks indicate owned data page pointers.

The transition from the left of the figure (i.e. the initial state) to the middle of the figure shows the a write to pages 3 and 8. The write to page 3 requires a change to PITC I (AP). PITC I follows the new page write processing to add page 3 to the table. PITC reads unchanged information from page J and uses the drive page B to store the page. All future writes to page 3 in this PITC are handled without moving pages. The write to page 8 depicts the second case for writing to a page. Since PITC I already contains page 8, PITC I writes over that portion of the data in page 8. For this case, it exists on the drive page C.

The transition from the middle of the figure to the right of the figure (i.e. final state) shows the coalescing of PITC II and III. Snapshot coalescing involves removing older pages, respectively, while maintaining all the changes in both PITC. Both PITC contain pages 3 and 8. The process retains the newer pages from PITC II and frees the pages from PITC III, and it returns pages A and D to the free list.

Snapshot allocates data pages from the page pool to store free list and PITC table information. Control Page allocation sub-allocates the data pages to match the sizes needed by the objects.

A volume contains a page pointer for the top of the control page information. From this page all of the other information can be read.

Snapshot tracks the number of pages in-use at certain time intervals. This allows snapshot to predict when the user needs to add more physical disk space to the system to prevent snapshot from running out.

Data Progression

In one embodiment of the present invention, data progression (DP) is used to move data gradually to storage space of appropriate cost. The present invention allows a user to add drives when the drives are actually needed. This would significantly reduce the overall cost of the disk drives.

Data progression moves non-recently accessed data and historical snapshot data to less expensive storage. For non-recently accessed data, it gradually reduces the cost of storage for any page that has not been recently

accessed. It may not move the data to the lowest cost storage immediately. For historical snapshot data, it moves the read-only pages to more efficient storage space, such as RAID 5, and to the least expensive storage if the page is no longer accessible by a volume.

The other advantages of the data progression of the present invention include maintaining fast I/O access to data currently being accessed, and reducing the need to purchase fast but expensive disk drives.

In operation, data progression determines the cost of storage using the cost of the physical media and the efficiency of RAID devices that are used for data protection. Data progression also determines the storage efficiency and moves the data accordingly. For example, data progression may convert RAID 10 to RAID 5 devices to more efficiently use the physical disk space.

Data progression defines accessible data as data that can be read or written by a server at the current time. It uses the accessibility to determine the class of storage a page should use. A page is read-only if it belongs to a historical PITC. If the server has not updated the page in the most recent PITC, the page is still accessible.

Figure 17 illustrates one embodiment of accessible data pages in a data progression operation. The accessible data pages is broken down into the following categories:

- Accessible Recently Accessed – These are the active pages the volume is using the most.

- Accessible Non-recently accessed – Read-write pages that have not been recently used.
- Historical Accessible – Read-only pages that may be read by a volume -- Applies to snapshot volumes.
- Historical Non-Accessible – Read-only data pages that are not being currently accessed by a volume -- Applies to snapshot volumes. Snapshot maintains these pages for recovery purposes, and the pages are generally placed on the lowest cost storage possible.

In Figure 17, three PITC with various owned pages for a snapshot volume are illustrated. A dynamic capacity volume is represented solely by PITC C. All of the pages are accessible and read-write. The pages may have different access time.

The following table illustrates various storage devices in an order of increasing efficiency or decreasing monetary expense. The list of storage devices may also follow a general order of slower write I/O access. Data progression computes efficiency of the logical protected space divided by the total physical space of a RAID device.

Table 1: RAID Types

Type	Sub Type	Storage Efficiency	1 Block Write I/O Count	Usage
RAID 10		50%	2	Primary Read-Write Accessible Storage with relatively good write performance.
RAID 5	3 – Drive	66.6%	4 (2 Read – 2 Write)	Minimum efficiency gain over RAID 10 while incurring the RAID 5 write penalty.

RAID 5	5 – Drive	80%	4 (2 Read – 2 Write)	Great candidate for Read-only historical information. Good candidate for non-recently accessed writable pages.
RAID 5	9 – Drive	88.8%	4 (2 Read – 2 Write)	Great candidate for read-only historical information.
RAID 5	17 – Drive	94.1%	4 (2 Read – 2 Write)	Reduced gain for efficiency while doubling the fault domain of a RAID device.

RAID 5 efficiency increases as the number of drives in the stripe increases. As the number of disks in a stripe increases, the fault domain increases. The increasing the numbers of drives in a stripe also increases the minimum number of disk necessary to create the RAID devices. In one embodiment, data progression does not use a RAID 5 stripe size larger than 9 drives due to the increase in the fault domain size and the limited efficiency increase. Data progression uses RAID 5 stripe sizes that are integer multiple of the snapshot page size. This allows data progression to perform full-stripe writes when moving pages to RAID 5 making the move more efficient. All RAID 5 configurations have the same write I/O characteristic for data progression purpose. For example, RAID 5 on an 2.5 inch FC disk may not effectively use the performance of those disks well. To prevent this combination, data progression needs to support the ability to prevent a RAID Type from running on certain disk types. The configuration of data progression can also prevent the system from using RAID 10 or RAID 5 space.

The types of disks are shown in the following table:

Table 2: Disk Types

Type	Speed	Cost	Issues
2.5 Inch FC	Great	High	Very Expensive
FC 15 K RPM	Good	Medium	Expensive
FC 10 K RPM	Good	Good	Reasonable Price
SATA	Fair	Low	Cheap/Less Reliable

Data progression includes the ability to automatically classify disk drives that are relative to the drives within a system. The system examines a disk to determine its performance relative to the other disks in the system. The faster disks are classified in a higher value classification, and the slower disks are classified in a lower value classification. As disks are added to the system, the system automatically rebalances the value classifications of the disks. This approach handles both the systems that never change and the systems that change frequently as new disks are added. The automatic classification may place multiple drive types within the same value classification. If the drives are determined to be close enough in value, then they have the same value.

In one embodiment, a system contains the following drives:

High - 10K FC drive

Low - SATA drive

With the addition of a 15K FC drive, Data progression automatically reclassifies the disks and demotes the 10K FC drive. This results in the following classifications:

High - 15K FC drive

Medium - 10K FC drive

Low – SATA drive

In another embodiment, a system may have the following drive types:

High - 25K FC drive

Low - 15K FC drive

Accordingly, the 15K FC drive is classified as the lower value classification, whereas the 15K FC drive is classified as the higher value classification.

If a SATA drive is added to the system, Data progression automatically reclassifies the disks. This results in the following classification:

High - 25K FC drive

Medium - 15K FC drive

Low – SATA drive

Data progression may include waterfall progression. Typically, waterfall progression moves data to a less expensive resource only when the resource becomes totally used. The waterfall progression effectively maximizes the use of the most expensive system resources. It also minimizes the cost of the system. Adding cheap disks to the lowest pool creates a larger pool at the bottom.

The typical waterfall progression uses RAID 10 space and then a next of RAID space, such as RAID 5 space. This forces the waterfall to go directly to RAID 10 of the next class of disks. Alternatively, data progression may include mixed RAID waterfall progression as shown in Figure 24. This alternative data progression method solves the problem of maximizing disk space and performance and allows storage to transform into a more efficient form in the same disk class. This alternative method also supports the requirement that RAID 10 and RAID 5 share the total resource of a disk class. This may require configuring a fixed percentage of disk space a RAID level may use for a class of disks. Accordingly, the alternative data progression method maximizes the use of expensive storage, while allowing room for another RAID class to coexist.

The mixed RAID waterfall also only moves pages to less expensive storage when the storage is limited. A threshold value, such as a percentage of the total disk space, limits the amount of storage of a certain RAID type. This maximizes the use of the most expensive storage in the system. When a storage approaches its limit, data progression automatically moves the pages to lower cost storage. Data progression may provide a buffer for write spikes.

It is appreciated that the above waterfall methods may move pages immediately to the lowest cost storage as in some cases, there may be a need in moving historical and non-accessible pages onto less expensive storage in a timely fashion. Historical pages may also be instantly moved to less expensive storage.

Figure 18 illustrates a flow chart of data progression process 1800. Data progression continuously checks each page in the system for its access pattern and storage cost to determine whether there are data pages to move. Data progression may also determine if the storage has reached its maximum allocation.

Data progression process determines if the page is accessible by any volume. The process checks PITC for each volume attached to a history to determine if the page is referenced. If the page is actively being used, the page may be eligible for promotion or a slow demotion. If the page is not accessible by any volume, it is moved to the lowest cost storage available. Data progression also factors in the time before a PITC expires. If snapshot schedules a PITC to expire shortly, no pages progress. If the page pool is operating in an aggressive mode, the pages may progress.

Data progression recent access detection needs to eliminate a burst of activity from promoting a page. Data progression separates read and write access tracking. This allows data progression to keep data on RAID 5 devices that are accessible. Operations like a virus scan or reporting only read the data. Data progression changes the qualifications of recent access when storage is running low. This allows data progression to more aggressively demote pages. It also helps fill the system from the bottom up when storage is running low.

Data progression may aggressively move data pages as system resources become low. More disks or a change in configuration are still necessary for all of these cases. Data progression lengthens the amount of time that the system may

operate in a tight situation. Data progression attempts to keep the system operational as long as possible. The time is when all of its storage classes are out-of-space.

In the case where RAID 10 space is running low, and total available disk space is running low, data progression may cannibalize RAID 10 disk space to move to more efficient RAID 5. This increases the overall capacity of the system at the price of write performance. More disks are still necessary. If a particular storage class is completely used, data progression allows for borrowing on non-acceptable pages to keep the system running. For example, if a volume is configured to use RAID 10-FC for its accessible information, it may allocate pages from RAID 5-FC or RAID 10-SATA until more RAID10-FC space is available.

Data progression also supports compression to increase the perceived capacity of the system. Compression may only be used for historical pages that are not accessed, or as the storage of recovery information. Compression appears as another class of storage near the bottom of storage costs.

As shown in Figure 25, the page pool essentially contains a free list and device information. The page pool needs to support multiple free lists, enhanced page allocation schemes, and the classification of free lists. The page pool maintains a separate free list for each class of storage. The allocation schemes allows a page to be allocated from one of many pools while setting minimum or maximum allowed classes. The classification of free lists comes from the device configuration. Each free list provides its own counters for statistics gathering and

display. Each free list also provides the RAID device efficiency information for the gathering of storage efficiency stats.

In one embodiment, the device list may require the additional ability to track the cost of the storage class. The combination determines the class of the storage. This would occur if the user would like more or less granularity with the configured classes.

Figure 26 illustrates one embodiment of a high performance database where all accessible data only resides on 2.5 FC drives, even if it is not recently accessed. Non-accessible historical data is moved to RAID 5 fiber channel.

Figure 27 illustrates one embodiment of a MRI image volume where accessible storage is SATA RAID 10 and RAID 5 for this dynamic volume. If the image is not recently accessed, the image is moved to RAID 5. New writes then go to RAID 10 initially.Figure 19 illustrates one embodiment of a compressed page layout. Data progression implements compression by sub-allocating fixed sized data pages. The sub-allocation information tracks the free portions of the page, and the location of the allocated portions of the page. Data progression may not predict the efficiency of compression and may handle variable sized pages within its sub-allocation.

Compressed page may significantly impact CPU performance. For write access, a compressed page would require the entire page be decompressed and recompressed. Therefore, pages actively being accessed are not compressed, and returned to their non-compressed state. Writes may be necessary in conditions where storeage is extremely limited.

The PITC remap table points to the sub-allocation information and is marked to indicate the page that is compressed. Accessing a compressed page may require a higher I/O count than a non-compressed page. The access may require the reading of the sub-allocation information to retrieve the location of the actual data. The compressed data may be read from the disk and decompressed on the processor.

Data progression may require compression to be able to decompress parts of the entire page. This allows data progression read access to only decompress small portions of the page. The read-ahead feature of read cache may help with the delays of compression. A single decompression may handle a number of server I/O. Data progression marks pages that are not good candidates for compression so that it does not continually attempt to compress a page.

Figure 20 illustrates one embodiment of data progression in a high level disk drive system in accordance with the principles of the present invention. Data Progression does not change the external behavior of a volume or the operation of the data path. Data progression may require modification to a page pool. The page pool essentially contains a free list and device information. The page pool needs to support multiple free lists, enhanced page allocation schemes, and the classification of free lists. The page pool maintains a separate free list for each class of storage. The allocation schemes allows a page to be allocated from one of many pools while setting minimum or maximum allowed classes. The classification of free lists may come from the device configuration. Each free list provides its own counters for statistics gathering and display. Each free list also

provides the RAID device efficiency information for the gathering of storage efficiency statistics.

The PITC identifies candidates for movement and blocks I/O to accessible pages when they move. Data progression continually examines the PITC for candidates. The accessibility of pages continually changes due to server I/O, new snapshot page updates, and view volume creation/deletion. Data progression also continually checks volume configuration changes and summarize the current list of page classes and counts. This allows data progression to evaluate the summary and determine if there are possibly pages to be moved.

Each PITC presents a counter for the number of pages used for each class of storage. Data progression uses this information to identify a PITC that makes a good candidate to move pages when a threshold is reached.

RAID allocates a device from a set of disks based on the cost of the disks. RAID also provides an API to retrieve the efficiency of a device or potential device. It also needs to return information on the number of I/O required for a write operation. Data progression may also require a RAID NULL to use third-party RAID controllers as a part of data progression. RAID NULL may consume an entire disk and merely act as a pass through layer.

Disk manager may also automatically determine and store the disk classification. Automatically determining the disk classification may require changes to SCSI Initiator.

From the above description and drawings, it will be understood by those of ordinary skill in the art that the particular embodiments shown and described

are for purposes of illustration only and are not intended to limit the scope of the present invention. Those of ordinary skill in the art will recognize that the present invention may be embodied in other specific forms without departing from its spirit or essential characteristics. References to details of particular embodiments are not intended to limit the scope of the invention.

CLAIMS

What is claimed is:

1. A disk drive system, capable of dynamically allocating data in a pool of storage, comprising:

a RAID subsystem having the pool of storage; and

a disk manager having at least one disk storage system controller;

wherein the RAID subsystem and the disk manager dynamically allocate data across the pool of storage and a plurality of disk drives based on RAID-to-disk mapping.

2. The system of claim 1, wherein the RAID subsystem and the disk manager determine whether additional disk drives are required, and a notification is sent if the additional disk drives are required.

3. The system of claim 1, wherein the disk manager manages a plurality of disk storage system controllers.

4. The system of claim 3, further comprising a plurality of redundant disk storage system controllers to cover failure of an operated disk storage system controller.

5. The system of claim 1, wherein the RAID subsystem further comprises a combination of at least one of RAID types, such as RAID-0, RAID-1, RAID-5, and RAID-10.

6. The system of claim 5, further comprising RAID types including RAID-3, RAID-4, RAID-6, and RAID-7.

7. The system of claim 1, wherein the pool of storage is a page pool of storage that maintains a free list of RAIDs.

8. The system of claim 1, wherein the pool of storage is a matrix of disk storage blocks that maintain a null list of RAIDs.

9. A method of dynamic data allocation, comprising the steps of:

defining a default size of disk space of a RAID subsystem which forms a pool of storage;

writing data;

allocating the data in the pool of storage;

determining occupancy rate of the disk space of the RAID subsystem based on historical occupancy rate of the disk space of the RAID subsystem;

determining whether additional disk drives are required; and

sending a notification to the RAID subsystem if the additional disk drives are required.

10. The method of claim 9, further comprising sending notification via an email.

11. The method of claim 9, further comprising setting a size of the disk storage block as a default and changeable by a user.

12. The method of claim 9, further comprising maintaining a free list of RAIDs, wherein the pool of storage is a page pool of storage that maintains the free list of RAIDs.

13. The method of claim 9, further comprising maintaining a null list of RAIDs, wherein the pool of storage is a matrix of disk storage blocks that maintain the null list of RAIDs.

14. A method of data instant replay, comprising the steps of:

defining a default size of disk space of a RAID subsystem which forms a pool of storage;

automatically generating a snapshot of the pool of storage at predetermined time intervals; and

storing an address index of the snapshot or delta in the pool of storage such that the snapshot or delta of the pool of storage can be instantly located via the stored address index.

15. The method of claim 14, wherein the step of automatically generating the snapshot of the RAID subsystem at the predetermined time intervals includes automatically generating a snapshot of the RAID subsystem at a user defined time interval.

16. The method of claim 15, wherein the time interval is in a range of every few minutes to every few hours.

17. The method of claim 14, wherein the snapshots are stored at a local RAID subsystem.

18. The method of claim 14, wherein the snapshots are stored at a remote RAID subsystem such that if a major system crash occurs, integrity of data is not affected, and the data can be instantly recovered.

19. The method of claim 14, wherein the pool of storage is a page pool of storage that maintains a free list of RAIDs.

20. The method of claim 14, wherein the pool of storage is a matrix of disk storage blocks that maintain a null list of RAIDs.

21. A system of data instant replay, comprising:

a RAID subsystem having a pool of storage; and

a disk manager having at least one disk storage system controller;

wherein the RAID subsystem and the disk manager dynamically allocate data across the pool of storage and a plurality of disk drives based on RAID-to-disk mapping.

22. The system of claim 21, further comprising a disk storage system controller for automatically generating a snapshot of the pool of storage at predetermined time intervals and storing an address index of the snapshot or delta in the pool of storage such that the snapshot or delta of the pool of storage can be instantly located via the stored address index.

23. The system of claim 22, wherein the disk storage system controller monitors frequency of data use from the snapshots of the pool of storage and applies an aging rule such that the less frequently used or accessed data is moved to the less expensive RAID subsystem.

24. The system of claim 23, wherein when data in the less expensive RAID subsystem starts to be used more frequently, the controller moves the data

to the more expensive RAID subsystem, thereby significantly reducing cost of the disk drive system.

25. The system of claim 21, wherein the pool of storage is a page pool of storage that maintains a free list of RAIDs.

26. The system of claim 21, wherein the pool of storage is a matrix of disk storage blocks that maintain a null list of RAIDs.

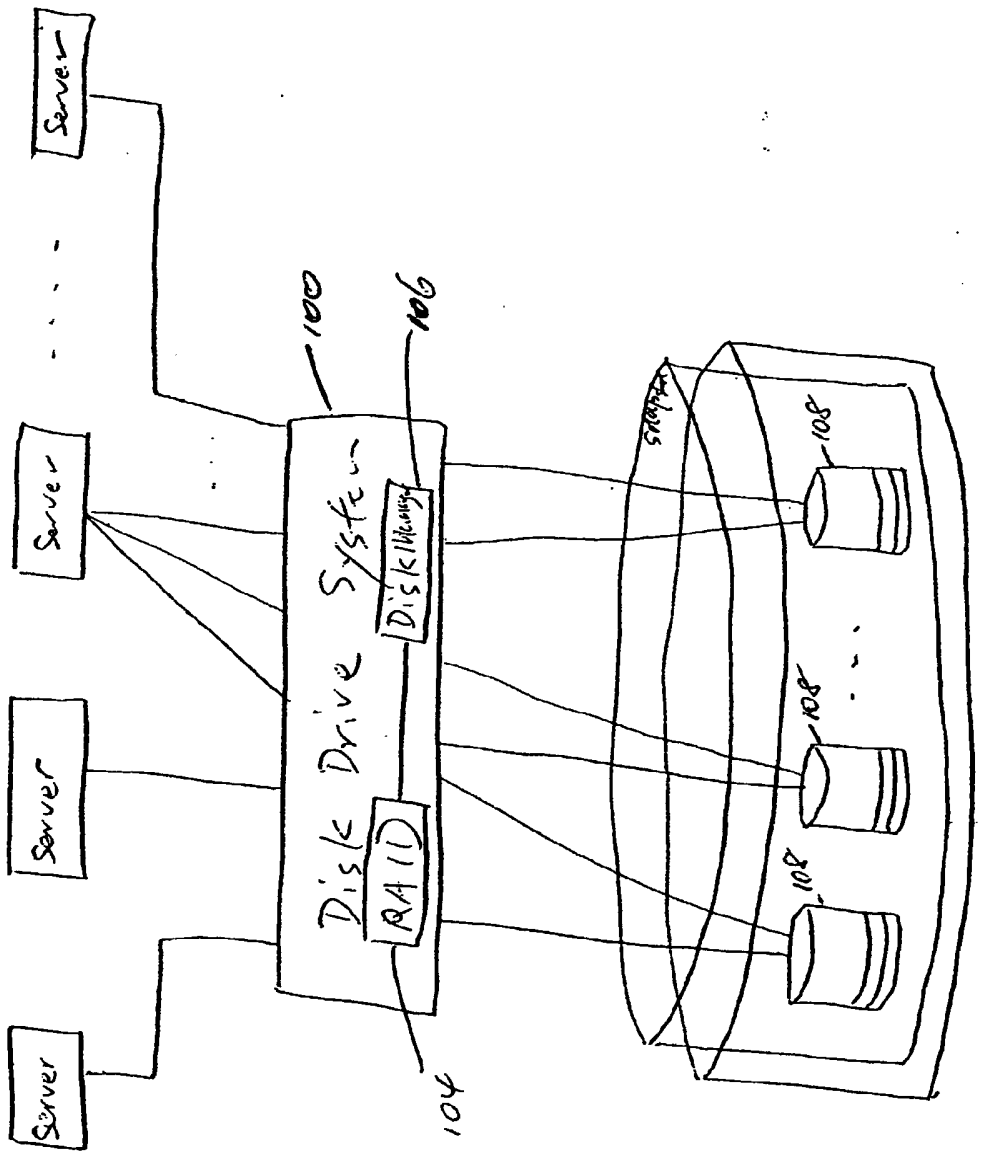

102
Server
Server
Server
Server
100
106
Disk Drive System
Disk Mirroring
RAID
104
Switch
108
108
108

Figure 1

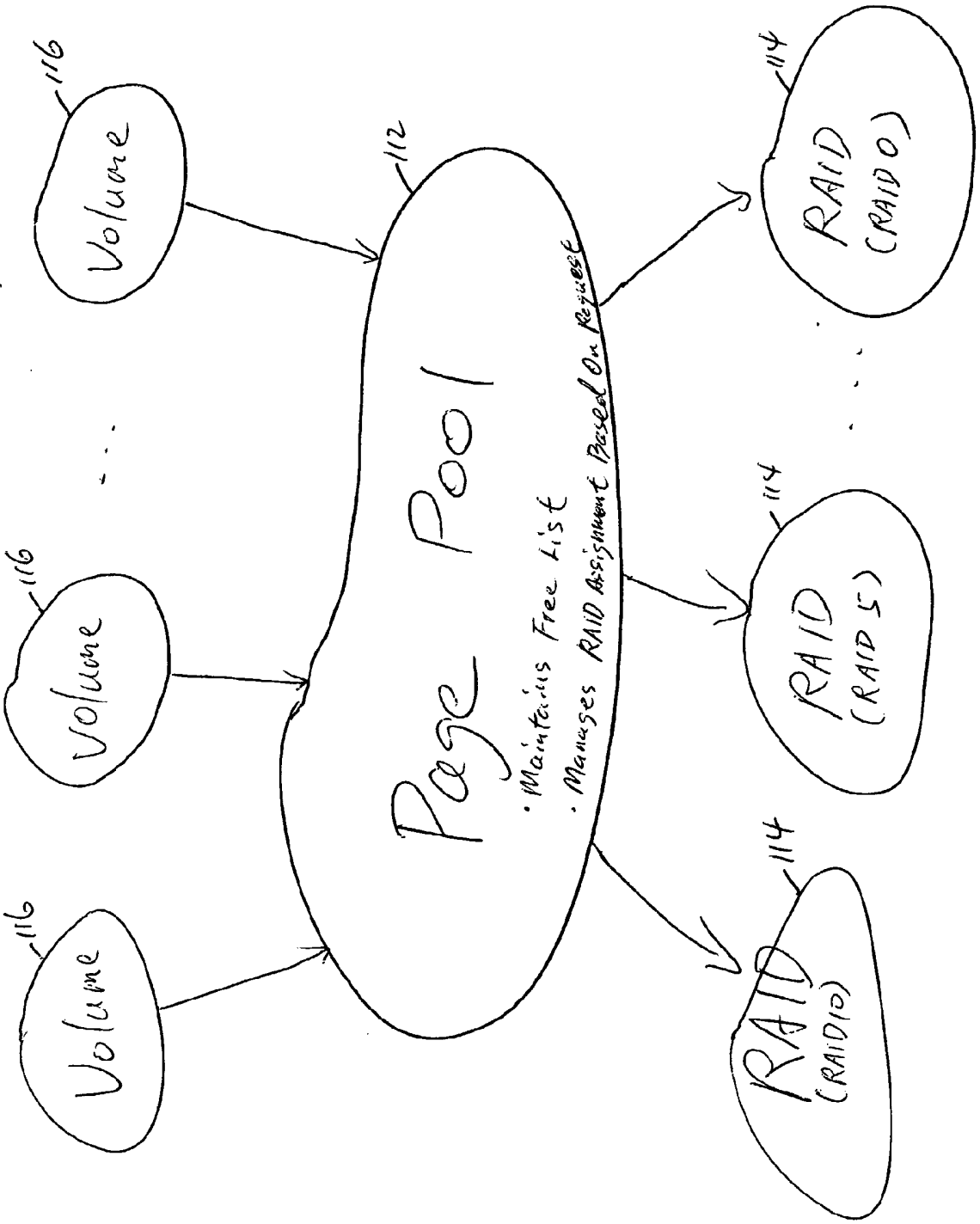

110
116
Volume
116
Volume
116
Volume
112
Page Pool
• Maintains Free List
• Manages RAID Assignment Based On Request
114
RAID
(RAID 0)
114
RAID
(RAID 5)
114
RAID
(RAID 10)

Figure 2

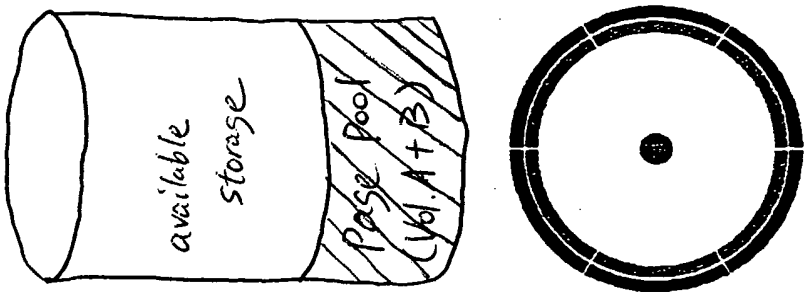


Figure 2B

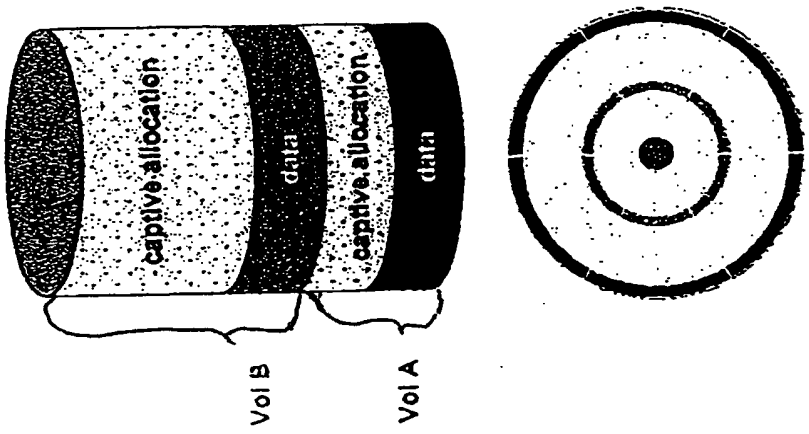


Figure 2A
(Prior Art)

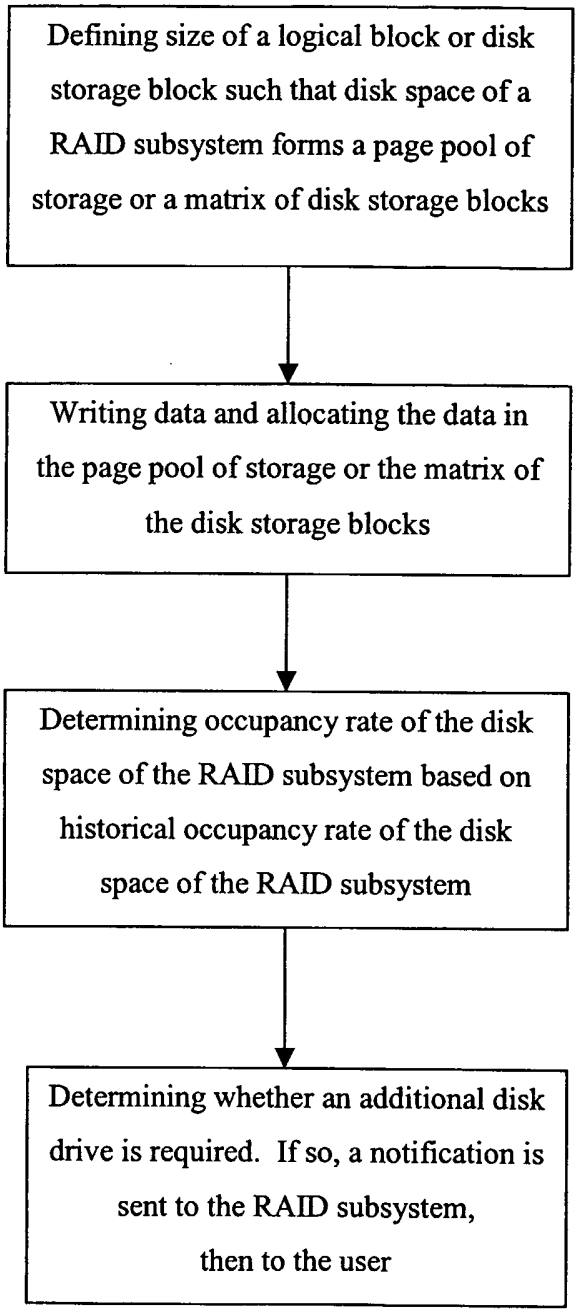

Defining size of a logical block or disk storage block such that disk space of a RAID subsystem forms a page pool of storage or a matrix of disk storage blocks
Writing data and allocating the data in the page pool of storage or the matrix of the disk storage blocks
Determining occupancy rate of the disk space of the RAID subsystem based on historical occupancy rate of the disk space of the RAID subsystem
Determining whether an additional disk drive is required. If so, a notification is sent to the RAID subsystem, then to the user

Figure 2C

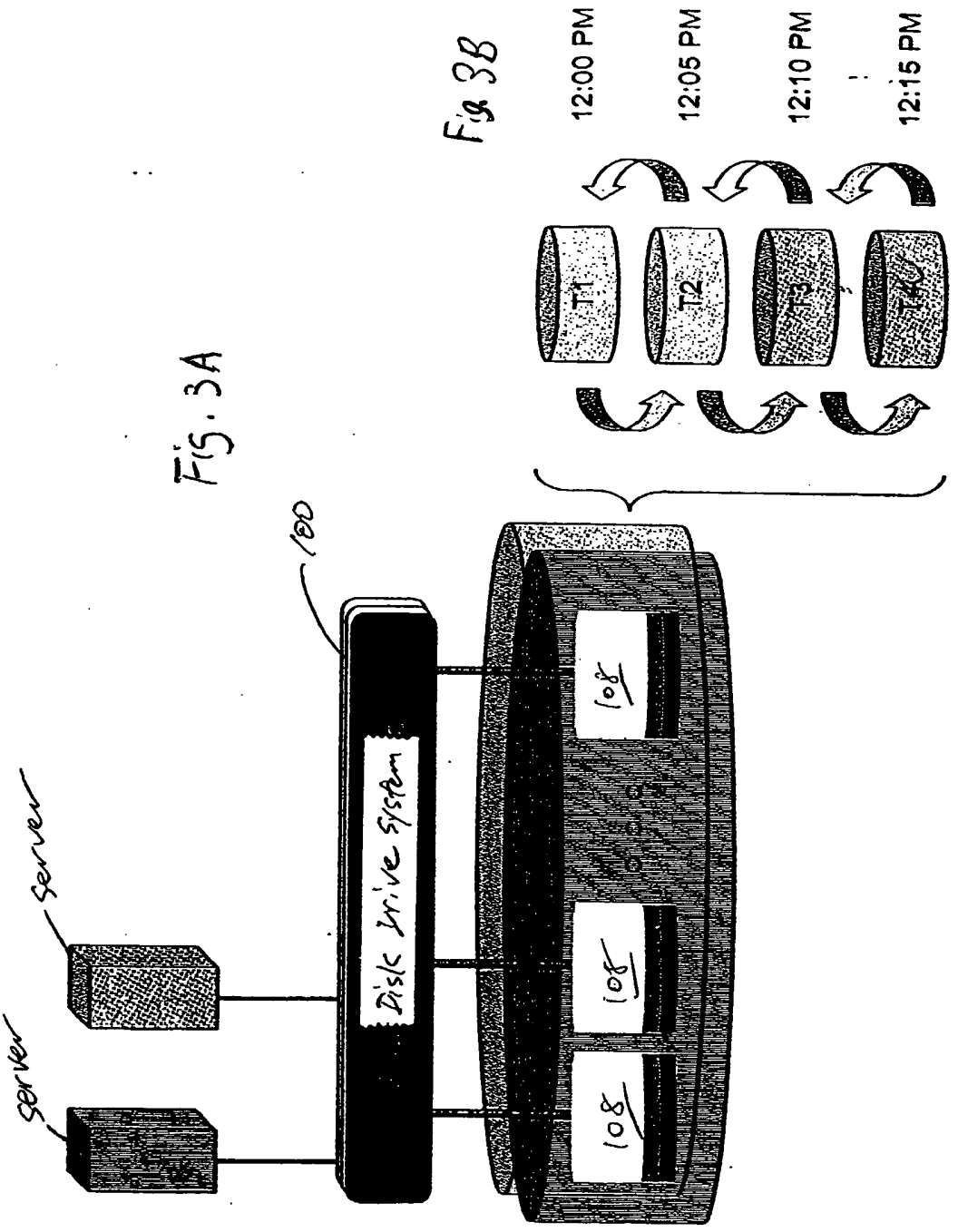


Fig 3A
Server
Server
Disk Drive System
100
108
108
108
Fig 3B
12:00 PM
12:05 PM
12:10 PM
12:15 PM
T1
T2
T3
TK

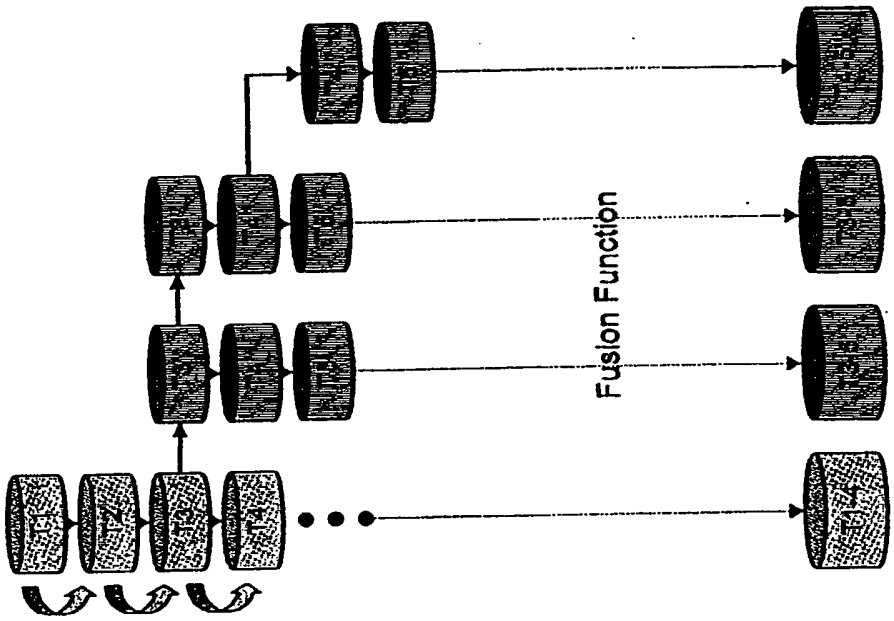


Fig. 4

500
Local Logical Volume
T1
T2
T3
T4
Remote Logical Volume
T1
T2
T3
T4
• Backup
• Reports
• Local failure
• Testing
• Virus or cyber intrusion
• SW failure/corruption
• Server corruption
• Centralized backup
• Vaulting
• Data center move
• Natural disaster
• Site disruption
• Power outage
Point to Point
Point to Multipoint
Peer to Peer

Fig. 5

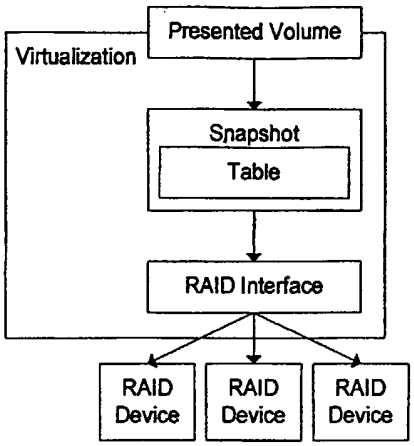

Virtualization
Presented Volume
Snapshot
Table
RAID Interface
RAID Device
RAID Device
RAID Device

Figure 6

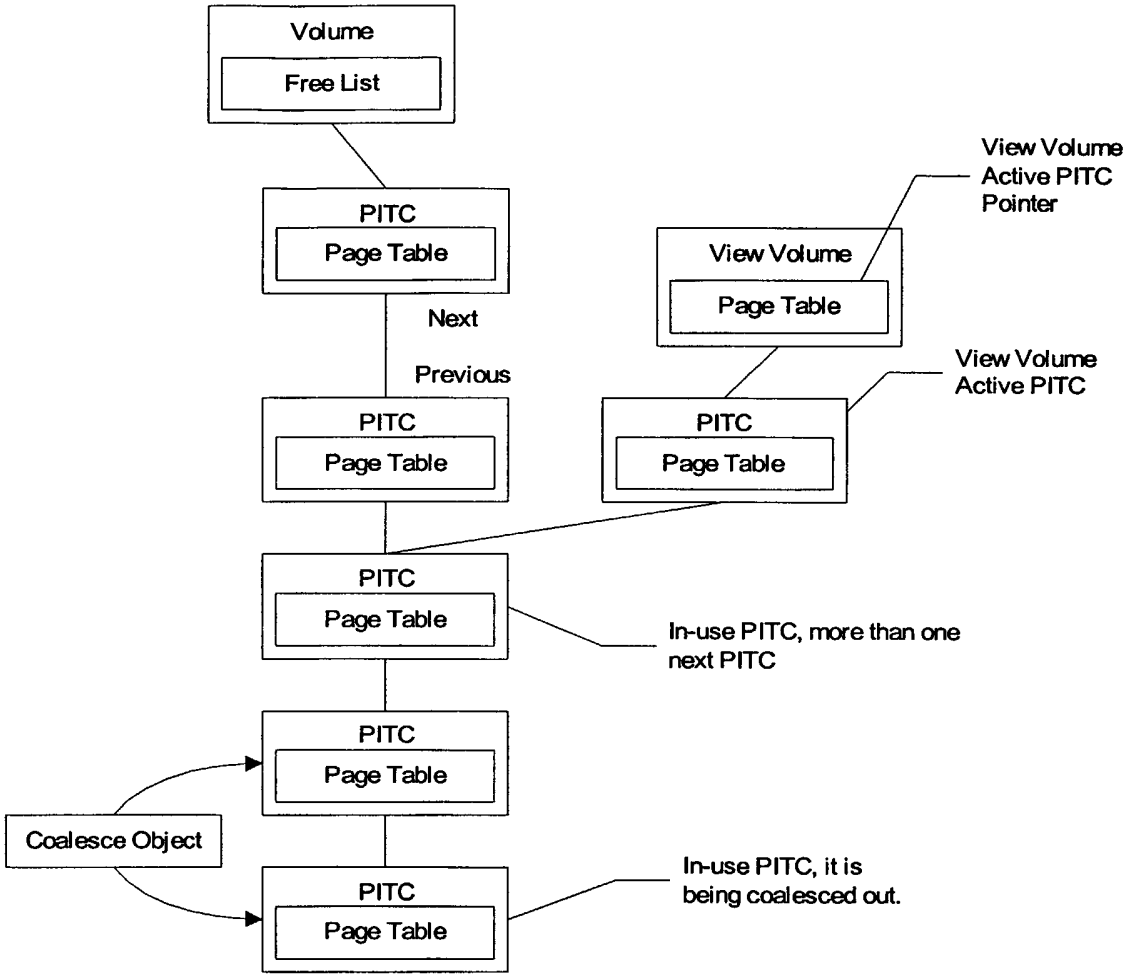

Volume
Free List
PITC
Page Table
Next
Previous
PITC
Page Table
PITC
Page Table
PITC
Page Table
PITC
Page Table
PITC
Page Table
View Volume
Page Table
View Volume
Active PITC
Pointer
View Volume
Active PITC
In-use PITC, more than one
next PITC
Coalesce Object
In-use PITC, it is
being coalesced out.

Figure 7

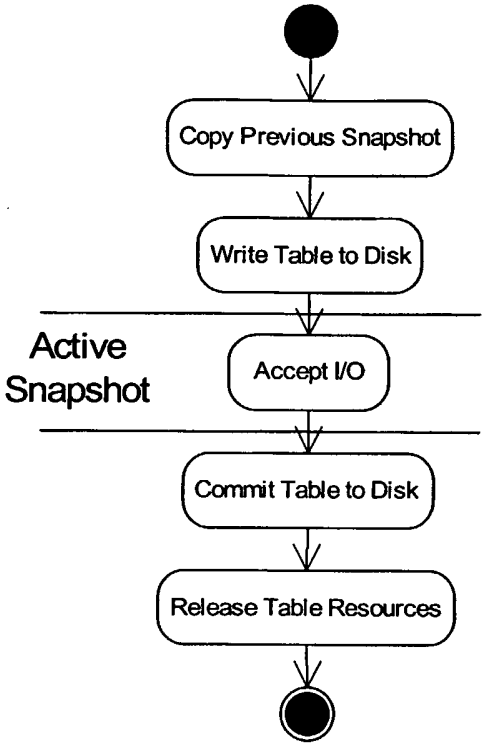

Copy Previous Snapshot
Write Table to Disk
Active Snapshot
Accept I/O
Commit Table to Disk
Release Table Resources

Figure 8

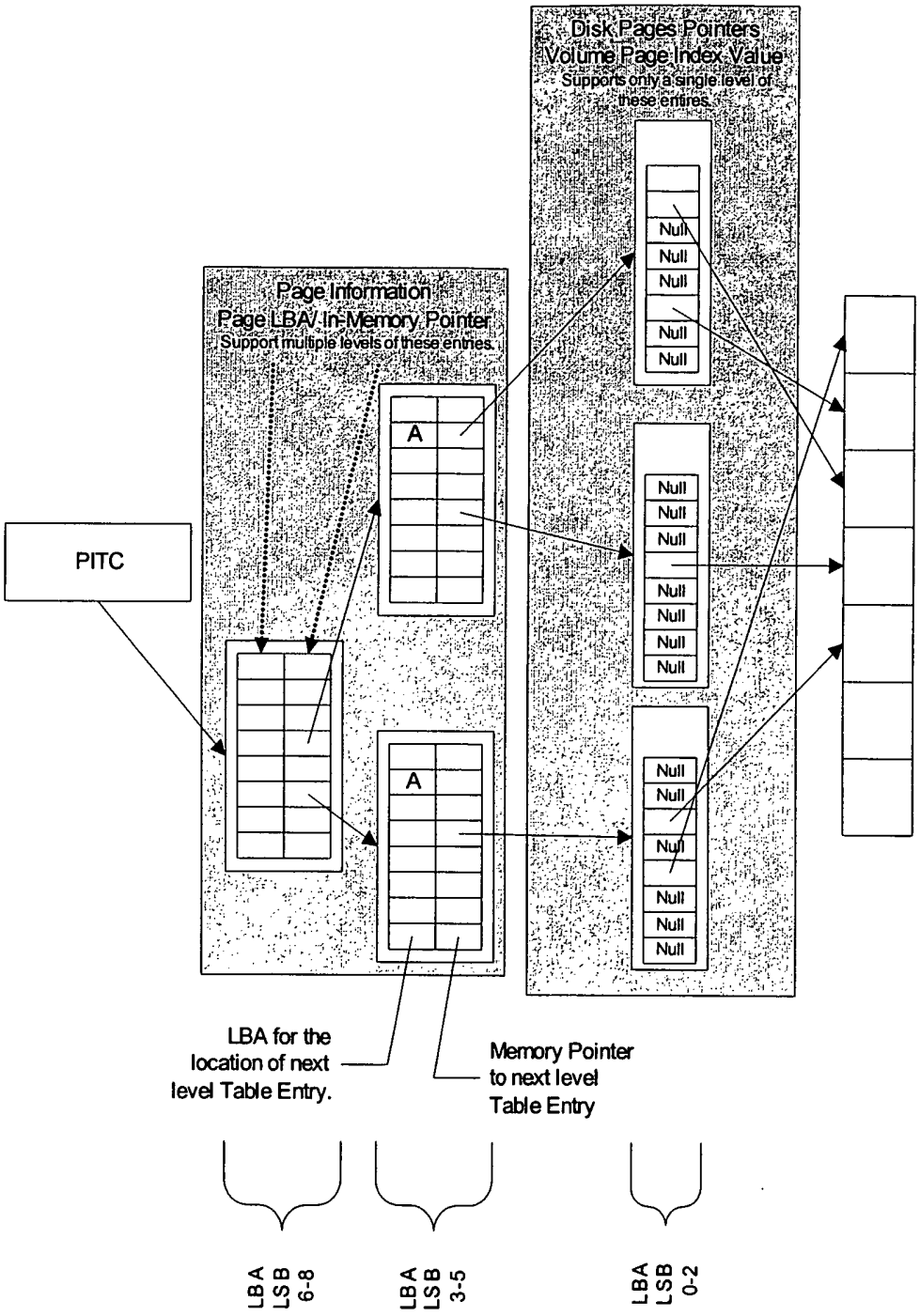

Disk Pages Pointers
Volume Page Index Value
Supports only a single level of these entires.
Null
Null
Null
Null
Null
Page Information
Page LBA/ In-Memory Pointer
Support multiple levels of these entries.
A
A
PITC
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
Null
LBA for the location of next level Table Entry.
Memory Pointer to next level Table Entry
LBA LSB 6-8
LBA LSB 3-5
LBA LSB 0-2

Figure 9

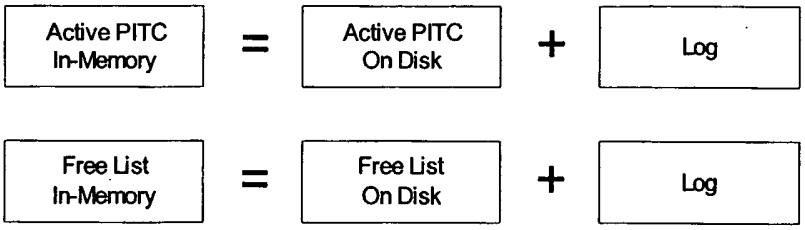

Active PITC In-Memory
=
Active PITC On Disk
+
Log
Free List In-Memory
=
Free List On Disk
+
Log

Figure 10

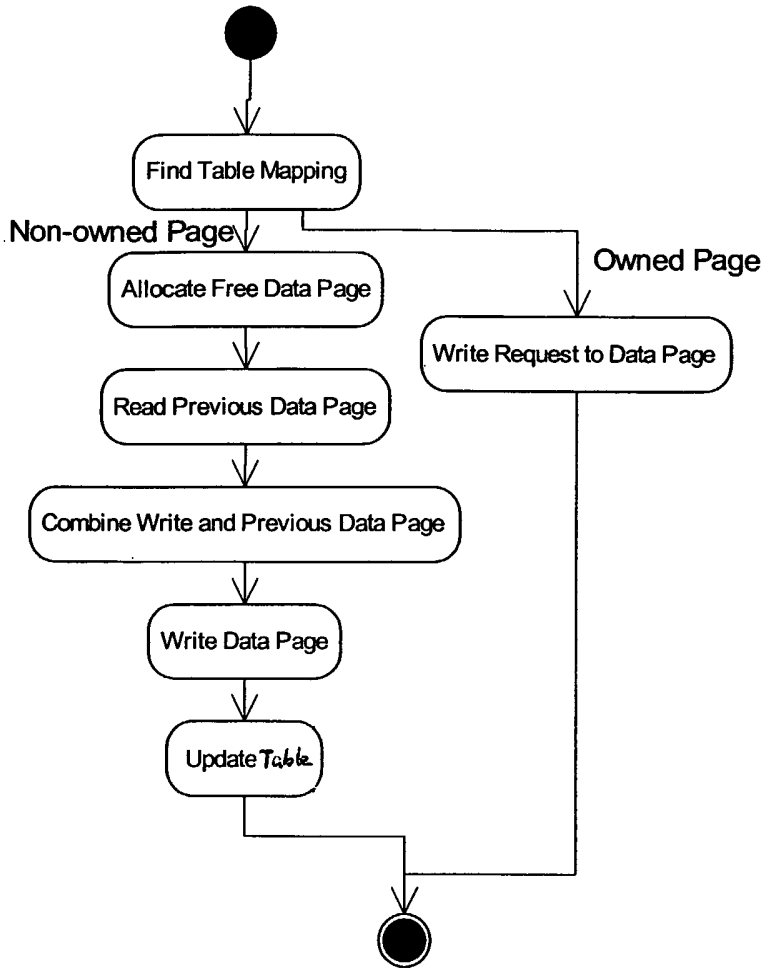

Find Table Mapping
Non-owned Page
Owned Page
Allocate Free Data Page
Write Request to Data Page
Read Previous Data Page
Combine Write and Previous Data Page
Write Data Page
Update Table

Figure 11

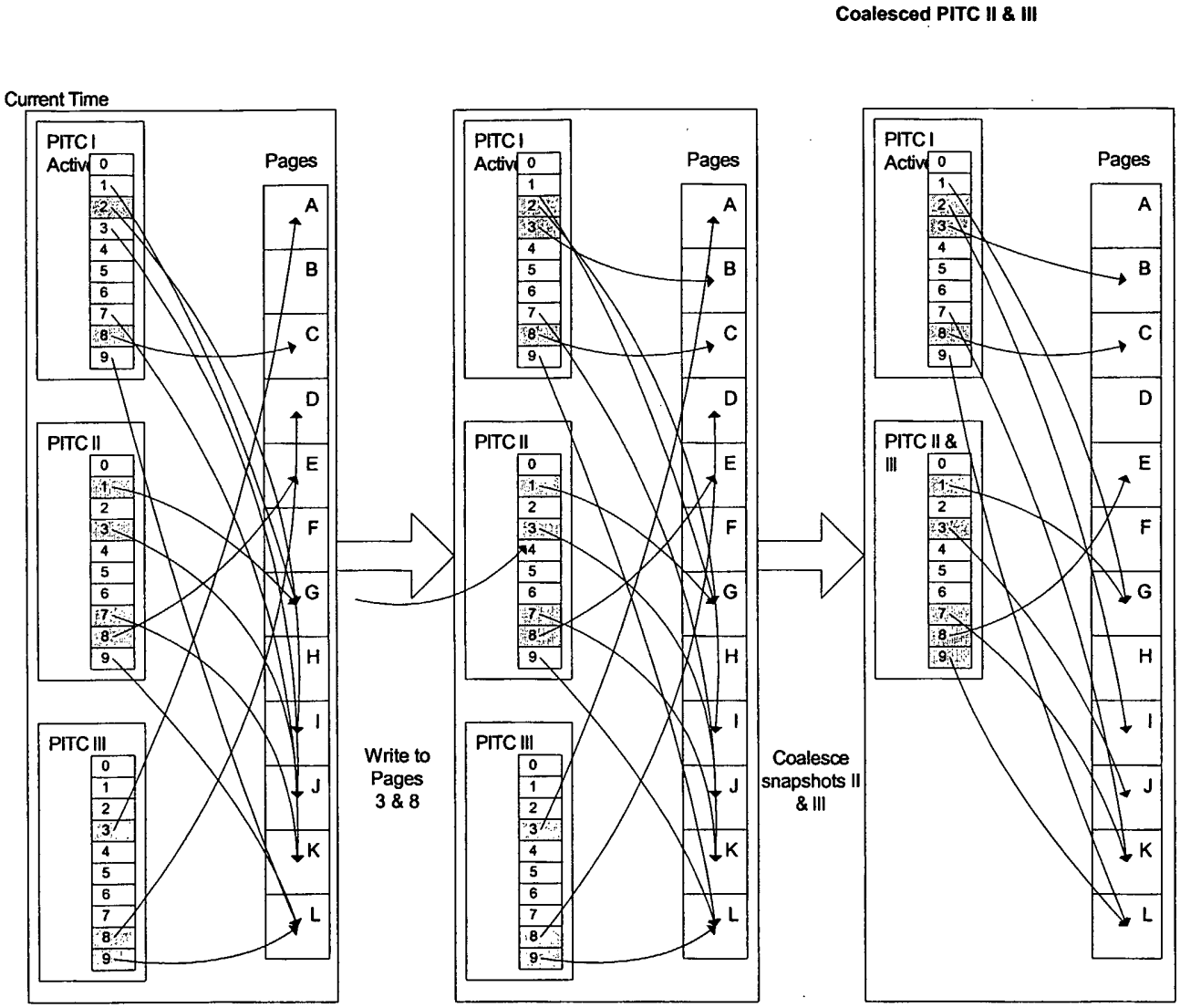

Coalesced PITC II & III
Current Time
PITC I
Pages
PITC II
PITC III
PITC II & III
A
B
C
D
E
F
G
H
I
J
K
L
Write to Pages 3 & 8
Coalesce snapshots II & III

Figure 12

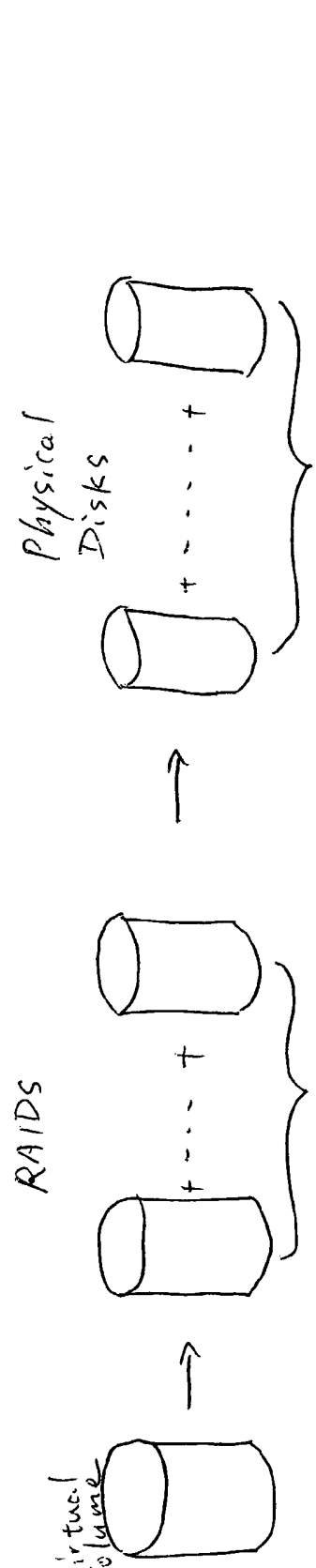


Figure 13A (Prior Art)

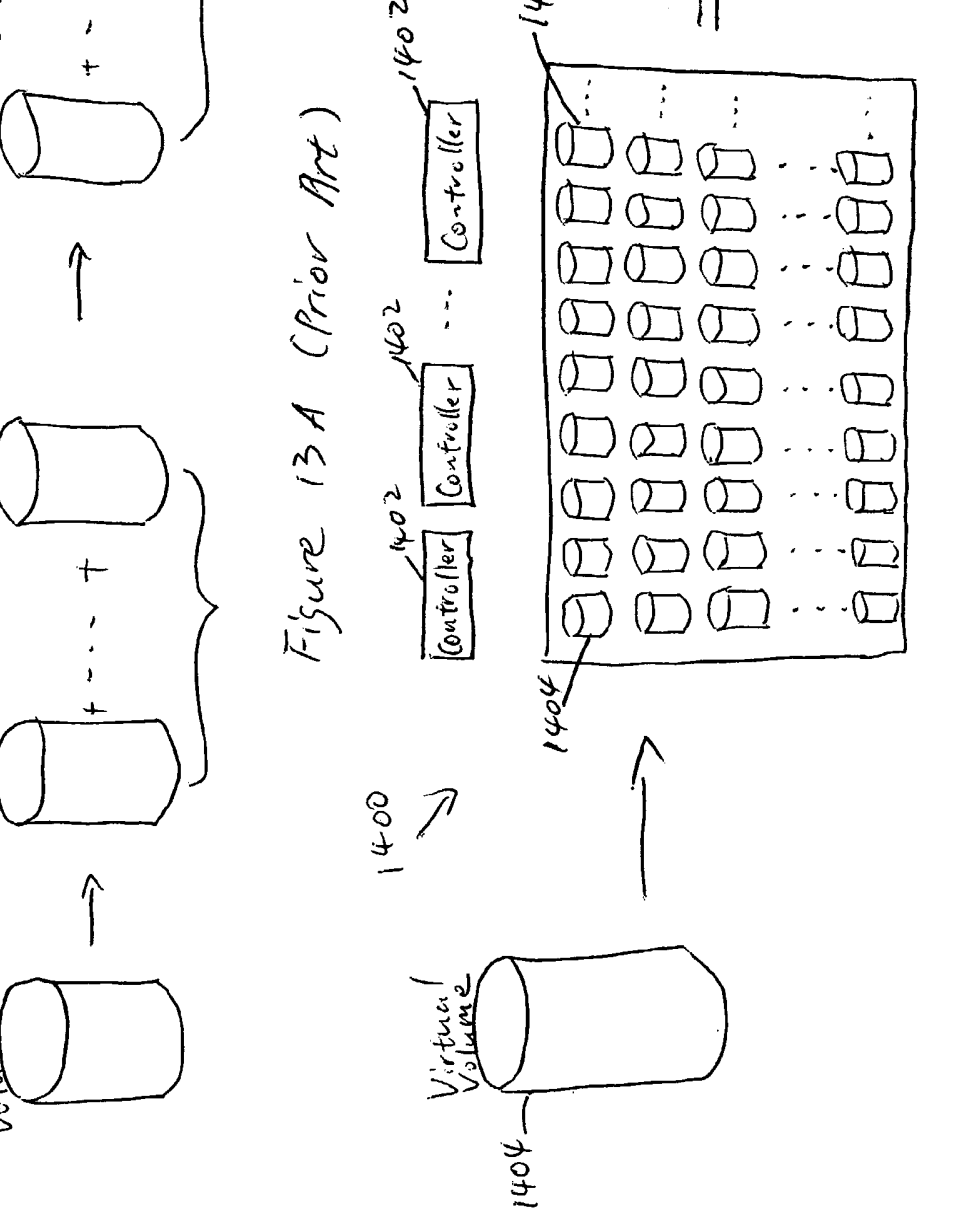


Figure 14A

Figure 14B

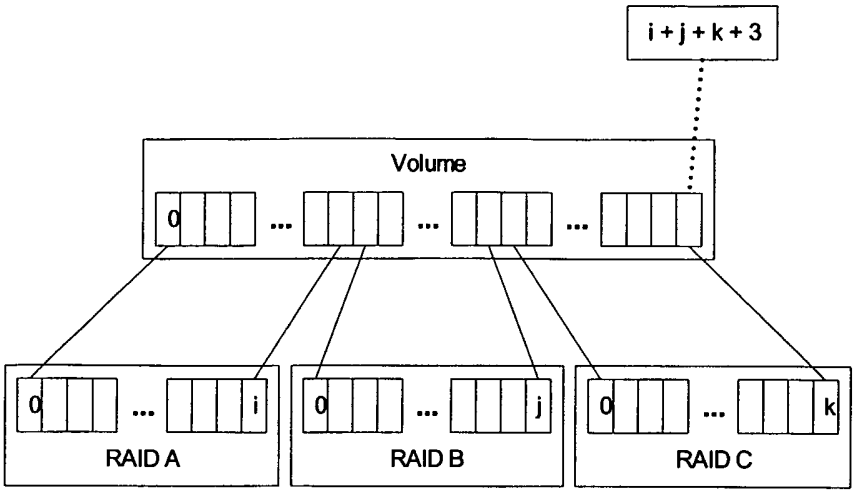


Figure 13B (Prior Art)

Page #	0	1	2	3	4	5	...
RAID Device	1	2	1	3	null	3	...
RAID Page #	54	1	52	112	null	64	

Figure 14C

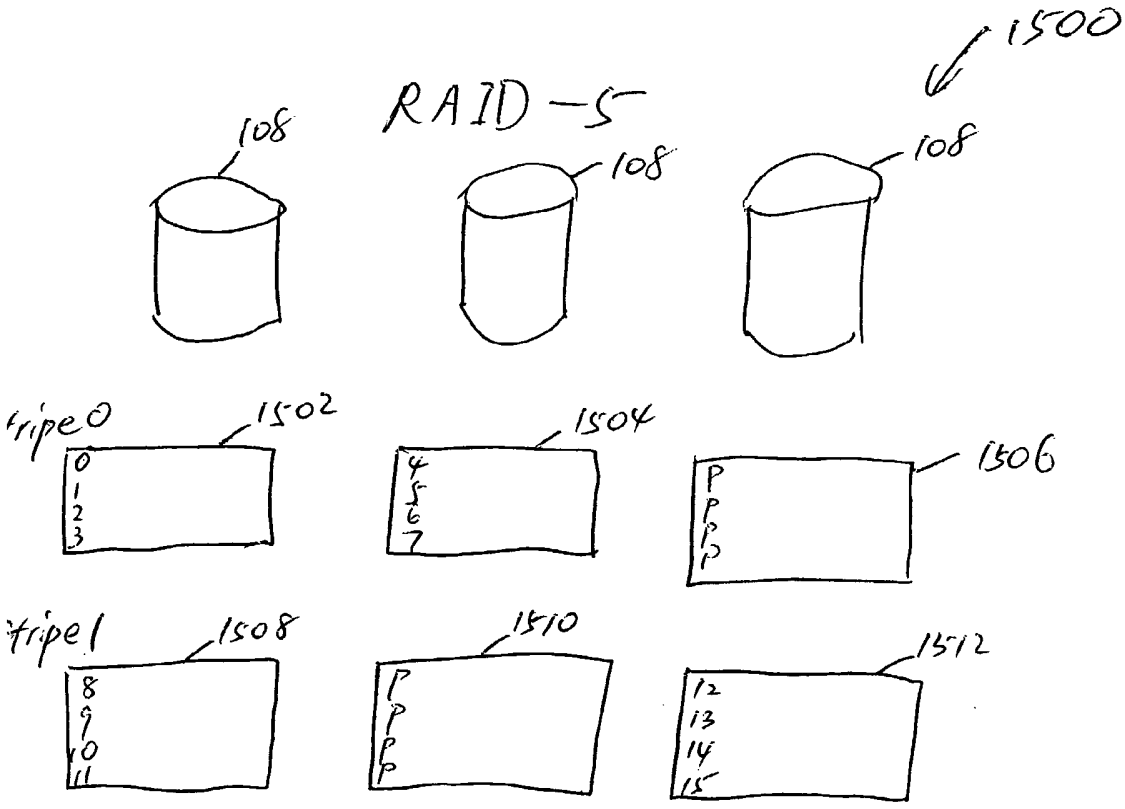


Figure 15

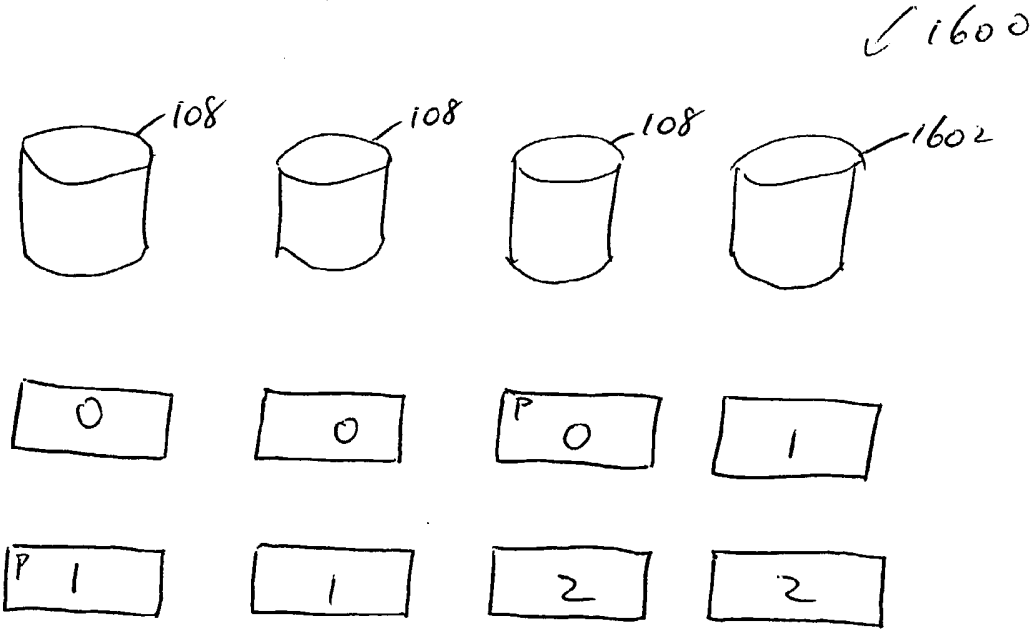


Figure 16

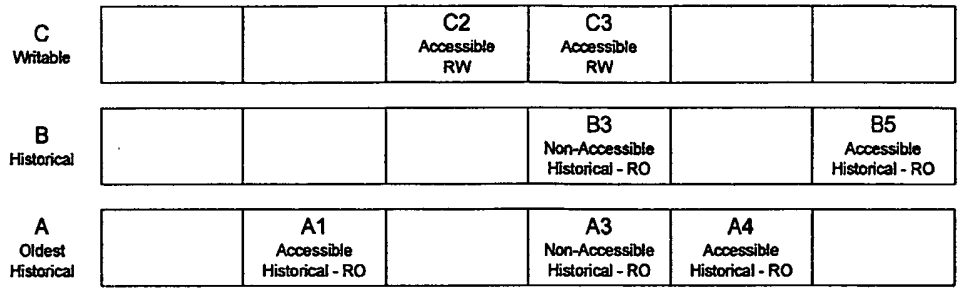

C
Writable
C2
Accessible
RW
C3
Accessible
RW
B
Historical
B3
Non-Accessible
Historical - RO
B5
Accessible
Historical - RO
A
Oldest
Historical
A1
Accessible
Historical - RO
A3
Non-Accessible
Historical - RO
A4
Accessible
Historical - RO

Figure 17

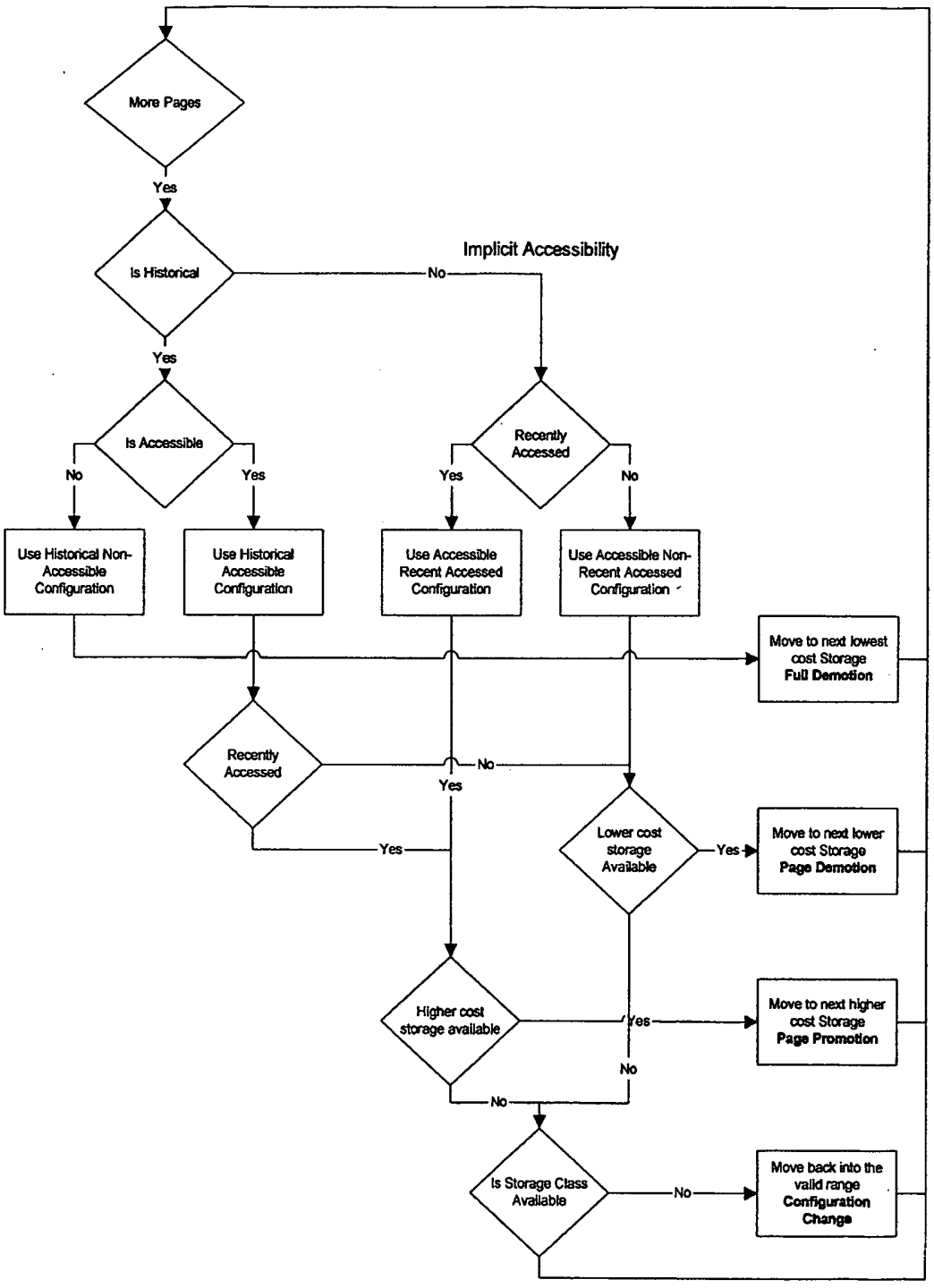

More Pages
Yes
Is Historical
Implicit Accessibility
No
Yes
Is Accessible
Recently Accessed
No
Yes
Yes
No
Use Historical Non-Accessible Configuration
Use Historical Accessible Configuration
Use Accessible Recent Accessed Configuration
Use Accessible Non-Recent Accessed Configuration
Move to next lowest cost Storage Full Demotion
Recently Accessed
No
Yes
Yes
Lower cost storage Available
Yes
Move to next lower cost Storage Page Demotion
Higher cost storage available
Yes
Move to next higher cost Storage Page Promotion
No
No
Is Storage Class Available
No
Move back into the valid range Configuration Change

Figure 18

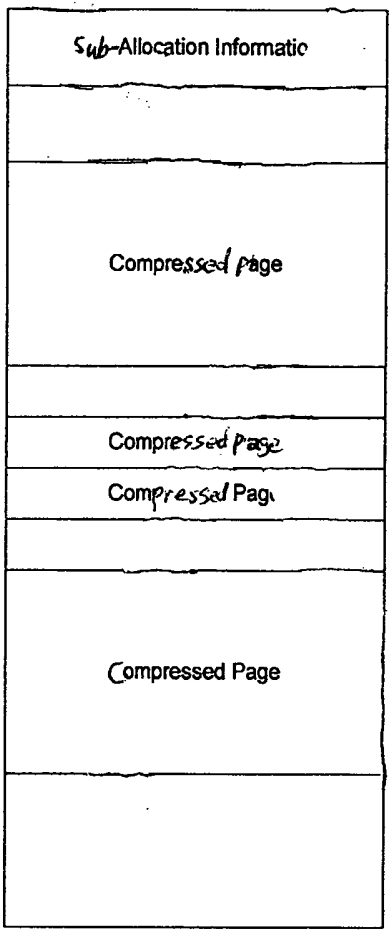

Sub-Allocation Informatio
Compressed Page
Compressed Page
Compressed Pag
Compressed Page

Figure 19

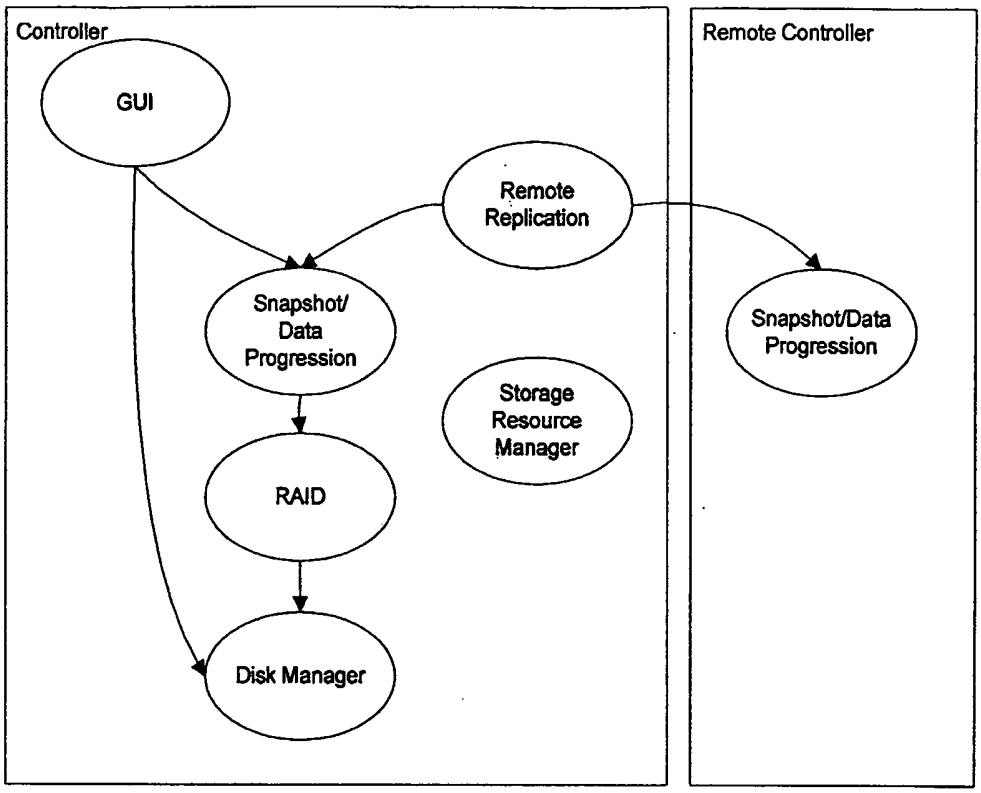

Controller
GUI
Remote Replication
Snapshot/ Data Progression
Storage Resource Manager
RAID
Disk Manager
Remote Controller
Snapshot/Data Progression

Figure 20

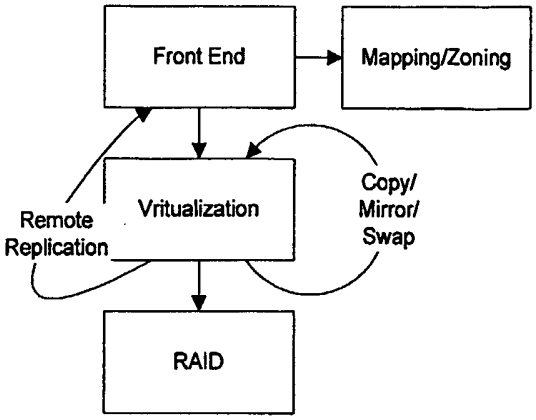

Front End
Mapping/Zoning
Vritualization
Remote
Replication
Copy/
Mirror/
Swap
RAID

Figure 21

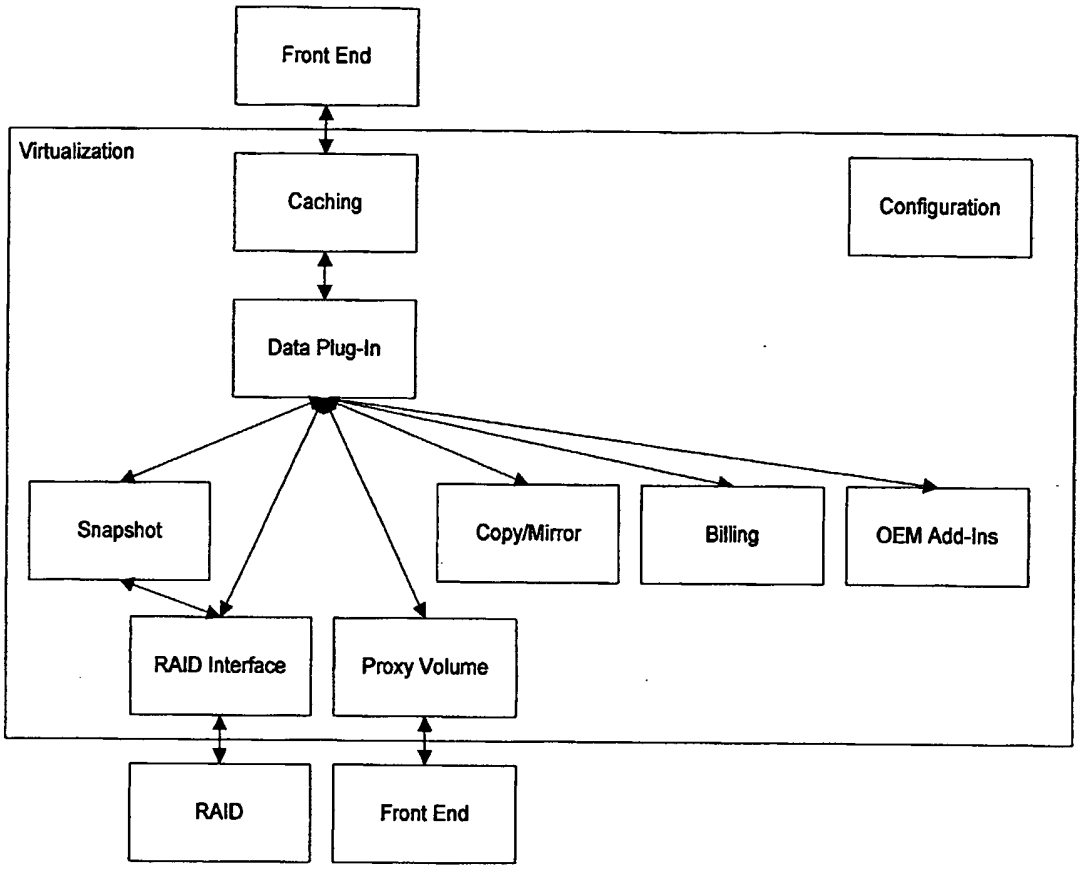

Front End
Virtualization
Caching
Configuration
Data Plug-In
Snapshot
Copy/Mirror
Billing
OEM Add-Ins
RAID Interface
Proxy Volume
RAID
Front End

Figure 22

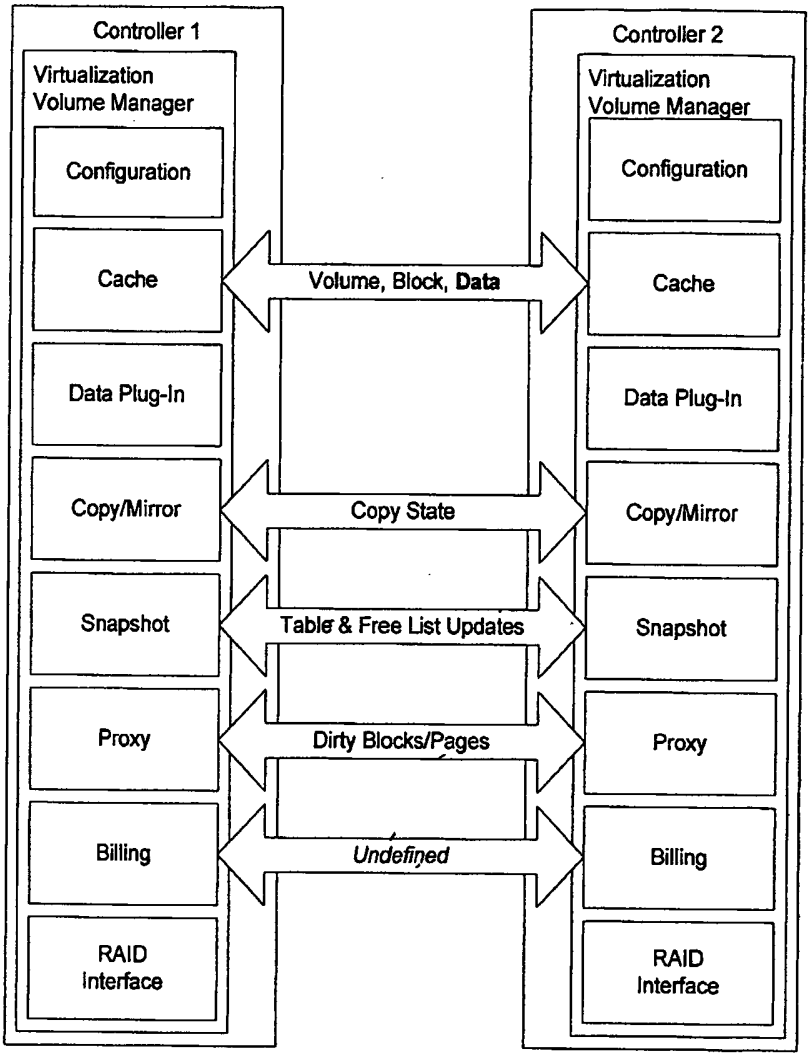

Controller 1
Virtualization Volume Manager
Configuration
Cache
Data Plug-In
Copy/Mirror
Snapshot
Proxy
Billing
RAID Interface
Controller 2
Virtualization Volume Manager
Configuration
Cache
Data Plug-In
Copy/Mirror
Snapshot
Proxy
Billing
RAID Interface
Volume, Block, Data
Copy State
Table & Free List Updates
Dirty Blocks/Pages
Undefined

Figure 23

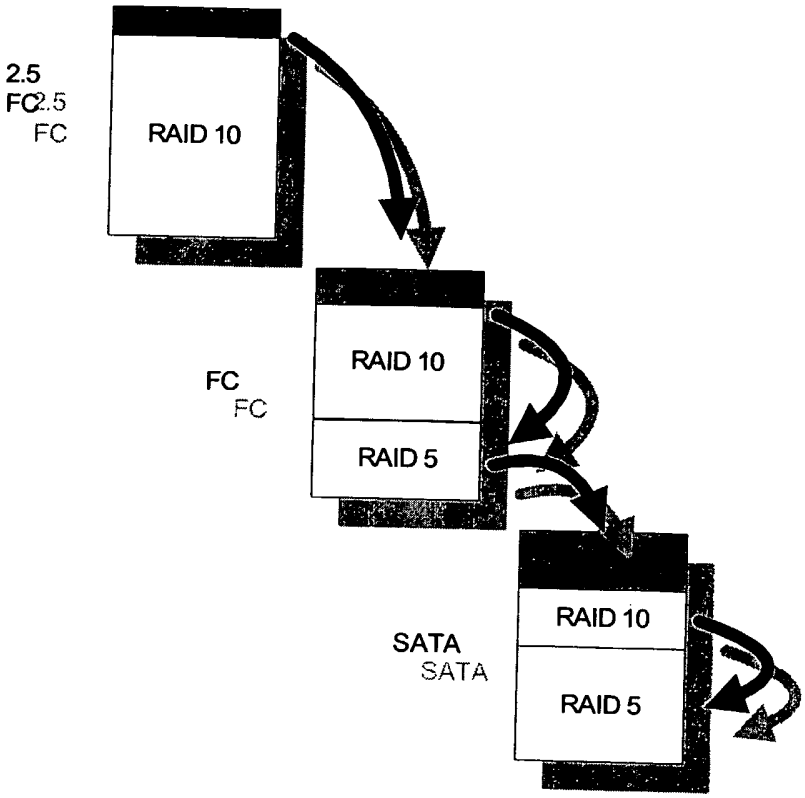


Figure 24

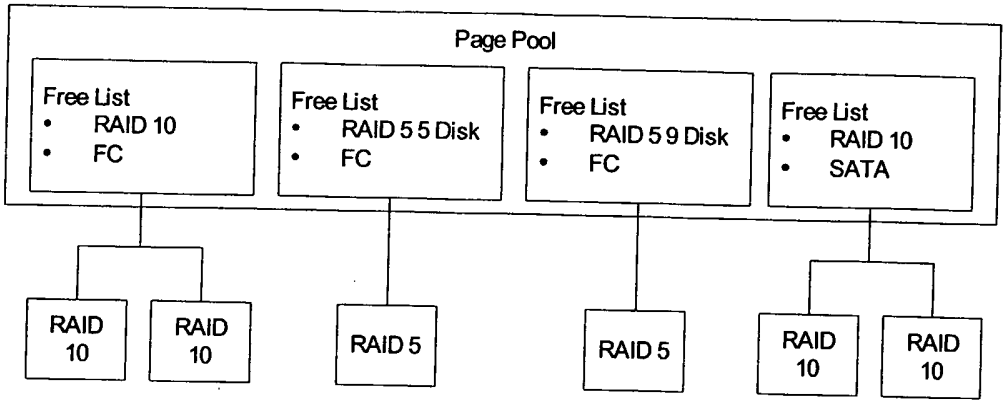

Page Pool
Free List
• RAID 10
• FC
Free List
• RAID 5 5 Disk
• FC
Free List
• RAID 5 9 Disk
• FC
Free List
• RAID 10
• SATA
RAID 10
RAID 10
RAID 5
RAID 5
RAID 10
RAID 10

Figure 25

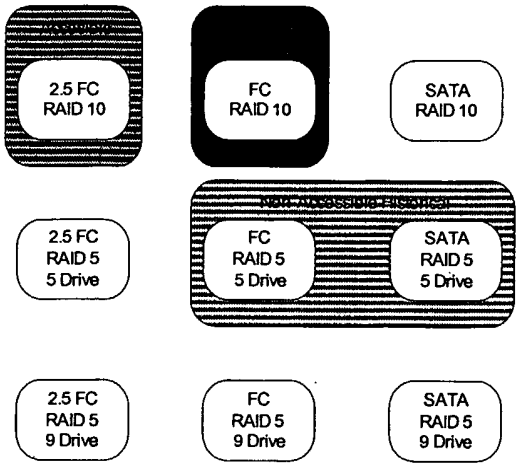

2.5 FC
RAID 10
FC
RAID 10
SATA
RAID 10
2.5 FC
RAID 5
5 Drive
FC
RAID 5
5 Drive
SATA
RAID 5
5 Drive
2.5 FC
RAID 5
9 Drive
FC
RAID 5
9 Drive
SATA
RAID 5
9 Drive

Figure 26

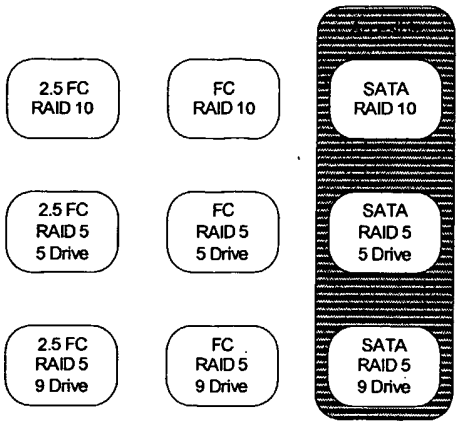

2.5 FC
RAID 10
FC
RAID 10
SATA
RAID 10
2.5 FC
RAID 5
5 Drive
FC
RAID 5
5 Drive
SATA
RAID 5
5 Drive
2.5 FC
RAID 5
9 Drive
FC
RAID 5
9 Drive
SATA
RAID 5
9 Drive

Figure 27