A Global Data Management Strategy for Distributed Computing Systems

Kingsley C. Nwosu,
IBM Large Scale Computing Div.,
MS/992, Neighborhood Rd.,
Kingston, NY 12401.

C. Y. Roger Chen,
Dept. Elect. & Comp. Engr.,
Syracuse University,
Syracuse, NY 13244.

P. Bruce Berra,
CASE Center,
Syracuse University,
Syracuse, NY, 13244.

Abstract

Due to the advancements in many facets of computing technologies and the immediate and changing needs of computer users and applications, distributed computing systems have enjoyed considerable attention in the past few years. Many distributed file systems and storage allocation techniques that have been developed for effective and reliable storage and retrieval of data have not fully exploited the capabilities available through Local Area Networks (LANs) and distributed environments. We present a Global Data Management Strategy (GDMS) for management of storage devices attached to machines that may be locally or remotely interconnected via a LAN. A distributed system model is presented and the functional characteristics of the GDMS, in terms of its space morphology and data availability techniques, are described. The operations for creation and maintenance of the storage space defined by the GDMS are also discussed.

Keywords: *disk striping, distributed file systems, distributed data management, logical storage space, object replication.*

1 Introduction

The information system explosions of the 1980's necessitated the integration of diverse system architectures and functions. Those changes were driven by users in response to their immediate and vast changing business requirements in conjunction with the need to improve the efficiency of their processing environments. Concomitant with the integration of these systems arose the necessity to share certain resources across different components of a distributed system. A *Distributed File System* (DFS) is responsible for controlling the storage and management of shared storage space and data in a distributed system. Several UNIX[1]-based DFSs [1]-[5] and high bandwidth data allocation techniques [6]-[8] have been designed and developed over the past several years. One of the primary objectives in the design and implementation of any DFS is *transparency*. Transparency mainly comes in the form of *network*, *mobility*, and *location*. Network transparency dictates that users be able to access remote files using the same file operations defined for local files. Mobility transparency allows a user to log in to any of the systems in the distributed environment and access required files. Location transparency hides the actual location of files from the users. It creates the impression, to the user, that his/her files are local to the operating site and that the name of an object does not have any relation to its physical location.

In spite of the advancements in network technology that have permitted the inter-connectivity and interoperatability of heterogeneous computing systems in a distributed environment, the configurability of storage devices that are remotely located or attached to different machines for defining logical storage spaces that are file system independent has not been fully, adequately, and aggressively exploited. Current technological capabilities, global information interaction, and user requirements necessitate that we be able to configure storage devices attached to several remote machines into a logical data management space. The area of Distributed Database Management (DDM) has contributed substantially in distributed data management. Many of the Distributed Data Management Systems (DDMSs) employ decomposition techniques for data fragmentation and storage in different machines that may be remotely connected via a network. However, the decomposition and storage of data are done with full cognizance of the data morphology. In the case of relational database systems, for example, the horizontal and vertical decompositions are performed on relation schemes on a set of tuples or at-

[1]UNIX is a registered trade mark of AT&T

tributes, respectively. In these techniques, it becomes imperative that each group of data be intrinsically associated with its location, therefore, naming convention becomes a vital issue.

In this paper, we present the design of a Global Data Management Strategy (GDMS) for storage space management on storage devices that may span remotely located computing systems. We present a review of the outstanding logical storage space management techniques that are utilized in UNIX-based systems. A distributed system model, on which the GDMS is based, is described. Since the storage space management strategy proposed is tailored towards distributed file systems, a review of the critical and important issues for distributed file systems are discussed. Some of these issues include the read/write operation semantics, servicing I/O requests, caching schemes, object replication, etc. An analysis of these issues, with respect to the GDMS, is presented. The processes and management interfaces for the creation, configuration, extension, and reduction of the logical space defined by the GDMS are described.

2 Logical Storage Space Techniques

In response to the limitations of the UNIX file systems, a number of logical storage space management techniques have been developed for managing multiple storage devices used to define a file system space. The original conventional UNIX file system was restricted to a disk drive or partition of a disk drive. In order to extend a full file system, a new file system was made on a bigger disk and the existing file system was dumped on the new disk. This problem was solved with the advent of logical data management sub-systems like the Logical Volume Manager (LVM) [10] and Disk Striping (DS) [11].

2.1 The Logical Volume Manager

The LVM is a pseudo-device driver that configures a number of disks or disk partitions into a logical data space and runs beneath a file system. The logical space created by an LVM is called a *Logical Volume* (LV). Each LV is represented by a UNIX special device file. File systems are created on LVs via the special device files. For example, Figure 1 shows an LV created from partition 2 of Disk 1, partitions 2 and 3 of Disk 2, partition 3 of Disk 3, and Disk 4. We assume that the their sizes are 1000, 500, 1000, 5000, and 10000 sectors, respectively. An allocation of 300 sectors from logical block 800 in the LV spans from partition 2 of Disk 1 to partition 2 of Disk 2. On the other hand, an allocation of 1000 blocks from logical block 1499 occupies partition 3 of Disk 2 entirely. The data space in an LV can be extended or reduced

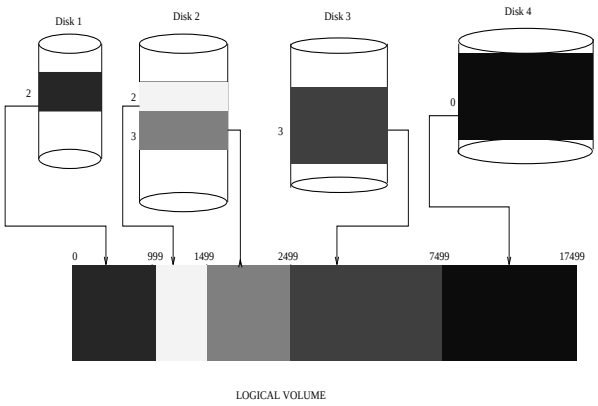

Figure 1: An example of a Logical Volume (LV).

by the system administrator by adding or removing disks or disk partitions. Besides logical data space management, the LVM also strives to mirror data for reliability and availability by keeping copies of data. For example, Figure 2 shows a sample storage device configuration for the LV in Figure 1. It shows a file system that was made on an LV that has its disk partitions from a number of disks that are attached to a single machine. The file system is then accessed by remote users via distributed file system capabilities.

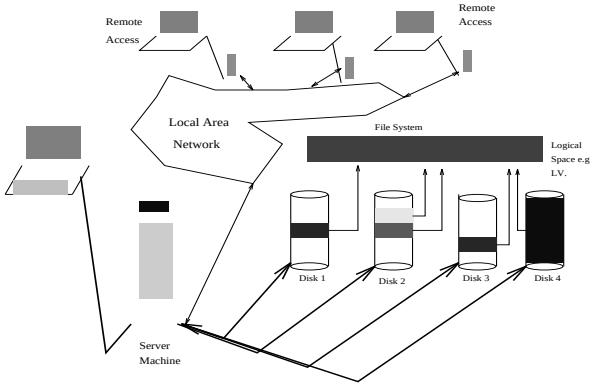

Figure 2: An example of a logical storage space configuration in LVM.

2.2 Disk Striping

The DS is also a pseudo-device driver that configures a number of disks or disk partitions into a contiguous logical data space. The storage devices are

logically arranged in two dimensional form and a specified number of blocks of data are allocated per storage device. A logical space created by the DS is called a *Disk Striping Device* (DSD). The number of storage devices in a DSD row (level) is called its *stripe width*. The devices in a row of a DSD must all be of the same size. The number of blocks allocated per allocation per device in a level is called the *stripe length*. For example, Figure 3 shows a DSD created from partition 2 of Disk 1, partition 3 of Disk 2, Disk 3, and Disk 4. Their sizes are 1000, 1000, 10000, and 10000 sectors, respectively. The stripe length is 10 blocks and the stripe width is 2. An allocation of 35 blocks starting from logical block 20 of the DSD will span physical blocks 10 to 19 of partition 2 of Disk 1, physical blocks 10 to 19 of partition 3 of Disk 2, physical blocks 20 to 29 of partition 2 of Disk 1, and physical blocks 20 to 24 of partition 3 of Disk 2. Furthermore, an allocation of 15 blocks starting from logical block 1995 of the DSD spans physical blocks 995 to 999 of partition 3 of Disk 2 and physical blocks 0 to 9 of Disk 3. The unit of extension or reduction of a DSD is a

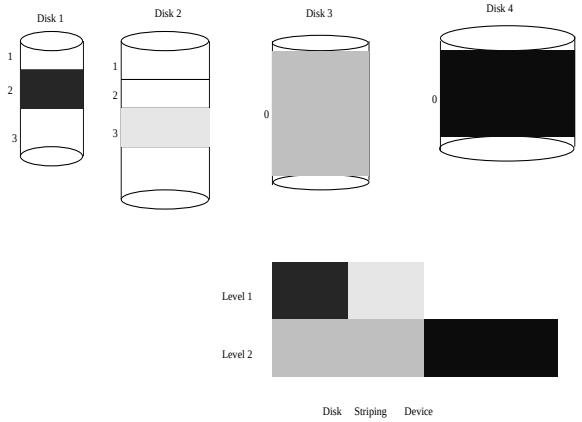

Figure 3: An example of a DSD.

level. A common property of pseudo-device drivers like the LVM or DS is the localization of the storage elements. The storage devices spanned by a DSD or an LV are usually attached to a single machine. A DFS is created on the logical space and mounted remotely on different machines so that remote users can access and manipulate the data.

3 Proposed Distributed System Model

The model of a distributed system proposed here is depicted in Figure 4. It consists of a number of remote machines (H_i) interconnected via a Local Area Network (LAN). Each machine in the network has a number of storage devices attached to it. A storage device attached to H_i is represented as S_k^i. The union of all the storage devices attached to all the sites in a network that are accessible is called the *Allocation Space* (AS). A subset of an AS is called an *Allocation Space Area* (ASA). A subset of an ASA whose members belong to one site is called an *Allocation Space Area-Segment* (ASAS). For example, a typical ASA, from Figure 4, may be $\{S_1^1, S_3^1, S_3^3, S_1^4\}$. In this case, the ASASs are $\{S_1^1, S_3^1\}, \{S_3^3\}$, and $\{S_1^4\}$. An ASA corresponds to the storage devices spanned by either an LV or a DSD. An ASAS is used to identify a set of storage devices of an ASA that belongs to a specific machine or site. The logical space spanned by an

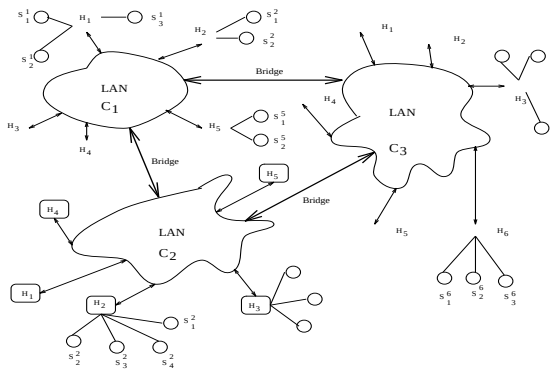

Figure 4: A sample of the proposed distributed system.

ASA is conceptually similar to that of the LV or DSD. However, unlike the LV or DSD, the constituent disks or disk partitions belong to different machines that are interconnected via a network. The logical space represented by an ASA is called an *Allocation Space Area Device* (ASAD). For example, using the storage device configuration of the LV shown in Figure 1, a typical ASAD can be similarly configured as shown in Figure 5.

4 Important Issues for Distributed Data Management

As is evident by now, GDMS is a network and de-centralization extension of the concept of logical storage space management and DFSs. We have reviewed the structural functionalities of some of the logical storage space management techniques such as the LVM and DS.

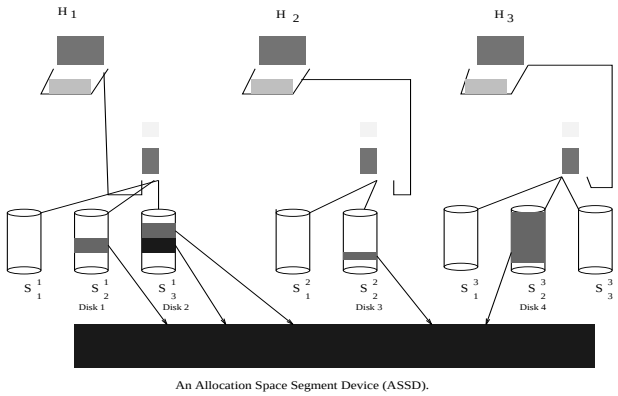

Figure 5: An example of an ASAD.

The bulk of the activities in our system model is concerned with distributed processing. As a result, in order to get a more vivid understanding of the issues that must be addressed in our system, we review some of the important issues and functionalities of distributed file systems and data management techniques. Although each distributed data management technique approaches a given problem differently, we will discuss the issues in general.

4.1 Read and Write Operations

The fundamental operations in a distributed data management system are the reading and writing of data. This operations are trivial by their nature, however, object sharing creates a lot of problems for these operations. The paramount concern in object sharing is the determination of the time at which a modified object becomes apparent to other clients accessing the object. In general, the temporal observability of changes in shared objects are file system dependent. A number of semantics for object sharing have been developed for different object management techniques. We present some synopses of some of those semantics.

UNIX Object Sharing Semantics: A client that reads an object must obtain the most current version of the object, i.e., the data resulting from the previous writes. A modification to an object should be immediately observable to other clients that concurrently and currently are accessing the object.

Session Semantics for Object Sharing: Stipulates that modifications to an open file are observable only to the initiating client while other clients that have the same file open are oblivious of any changes. The changes only become available to other clients that opened the file after the file has been closed.

Transaction-based Sharing Semantics: The result of a number of sessions is expected to be the same as if the sessions were executed in some controlled serial order. These semantics are usually accomplished by object locking.

4.2 Servicing I/O Requests

Since the data needed by a client in a distributed system may be locally or remotely situated, mechanisms exist that define how a desired operation by a client is carried out locally or remotely. The two prominent approaches are *external service* and *localized service*. External service is achieved when a client can only handle operations on data that are local to it. Localized service entails that the client obtains the desired data from a remote site, when necessary, and manipulates the data by itself.

4.3 Issues in Caching Schemes

The issue of caching has been around basically as a consequence of the disparity between the access speeds of the main memory and secondary storage devices; and main memory and processors. Classically, most people think of cache memories as high speed buffers which are inserted between the processors and main memory to capture those portions of the contents of the memory which are currently in use. Cache memories can also be inserted between main memory and secondary storage devices. In the following discussions, we assume the later. Some cache designs exist for faster accesses while others exist for localization of data. In distributed computing environments, the overriding motivation is to reduce the degree of intersite communications, whenever possible.

Cache Update Policies: The time when some data in the master copy is updated after a write depends on the write policy. One possibility, *write-through* (WT), is to update directly the master copy of the data as soon as it is written to any cache. Alternatively, writes to the master copy may be *delayed*. One aspect of this is called the *write-back* (WB) policy in which the master copy is written only when the cache block is replaced.

Cache Replacement Policies: The main objective of a replacement policy is to retain those cache blocks likely to be referenced in the near future and discard those that are no longer useful or whose next access is in the more distant future. A cache replacement strategy is used when the cache

is full. Various replacement algorithms have been proposed to select the block to replace. Another important goal of any block replacement algorithm is also to reduce the miss ratio.

Cache Coherence Issues: An important condition for cache replacement or fetch is the problem of determining whether or not a client's local cache copy is consistent with the master copy. A client must determine the validity of the copy of the data that it accesses in its local cache. If the local cache copy is out-dated, then the valid copy must be obtained from the server. In other to accomplish that, a client may employ one of the these two approaches:

- *Client-initiated approach*: Before a client utilizes any of its cache blocks, it contacts the server to determine if its local copy is consistent with the server's copy.
- *Server-initiated approach*: It is the responsibility of the server to notify every client that has a copy of its master copy about potential inconsistencies. A potential inconsistency exists whenever one of the clients modifies one of the copies of a master file.

4.4 Object Replication

Replication of storage in a distributed file system serves multiple purposes. First, from the user's point of view, multiple copies of data provide the opportunity for substantially increased availability. From the system point of view, some form of replication is more than convenient; it is absolutely essential for system data structures, both for availability and performance. In distributed computing environment, replication of data on different machines is more meaningful than on different media on the same machine. The overriding requirement for replication is that different replicas of the same object or part of an object reside on failure-independent machines. Above all, it is imperative that object replication be transparent to the users.

The inherent problem with replicated objects or data is their update. Logically, to the user, replicas of an object denote the same entity, consequently, an update to one replica must be reflected on all other replicas. Invariably, there is a trade-off between object availability and consistency. However, the consistency of objects cannot be compromised.

5 Problem Formulations

Although these and other related and equally important issues have been efficiently resolved in distributed file systems and data management strategies, however, the structural configuration of our GDMS increases the complexity of those issues. The primary objective of any data management technique is to guarantee data consistency, reliability, and availability. In view of these issues and other necessary concerns, the problems become: (1) What are the physical and management constituents for a logical storage space defined by the GDMS? (2) What are the space limitations for the GDMS storage space? (3) Given the physical and management components, how do we create, maintain, and manage the logical space? (4) What kinds and types of sharing semantics must exist between the GDMS and its clients (e.g., a file system)? (5) What type of I/O service protocols must exist between client and server sites? (6) What kind of cache update policy must exist between the GDMS and its clients? (7) What kind of data replication strategy will be employed and how are consistency and management strategies enforced for data replication? (8) What kind of inter-process communications exist between the client and server processes?

6 Functional Design of the GDMS

The Global Data Management Strategy (GDMS) presented here is an optional software component that is available to manage storage devices in UNIX-based systems. It provides block-oriented storage to any system component that requires it. A file system can be created and mounted on it and both read and write requests can be issued directly to an ASAD. A block storage device used for an ASAD is called a *Storage Volume* (SV). The storage space provided by an ASAD can be used by the virtual memory system, a file system, or an application that accesses "raw" block devices. Using an ASAD to manage disk space has the following advantages: (1) An ASAD may be larger than a single available disk drive. (2) Remote disks can be configured within an ASAD. (3) An ASAD can expand and contract at the discretion of the system administrator. (4) Data can be replicated in an ASAD for reliability and availability.

6.1 An ASAD Extent

As pointed out, the space spanned by an ASAD is not restricted within a single disk or a single site. The

SVs of an ASAD may be partitions of a single disk in a site, partitions of different disks in a site, or partitions of different disks in different sites within a given network environment. An ASAD can be extended and reduced, when necessary, by the system administrator. An extension merely involves the addition of one or more SVs to the ASA of the corresponding ASAD. An extension does not affect the objects already stored in an ASAD. Reducing an ASAD requires the removal of one or more SVs. It is obvious that reducing an ASAD may result in loss of valuable data. That decision is left to the system administrator. The GDMS appropriately warns the system administrator when loss of value data is imminent as a result of a reduction.

6.2 Storage Volume Mirroring

The GDMS allows data stored in a physical block of an SV to be replicated (mirrored) for high availability, reliability, and performance. SVs can be selectively mirrored in any number of copies (limited by system definition) at any time during the life of an ASAD. It is required that the mirrors of an SV be characteristically similar to the parent SV. This requirement is merely for ease of I/O re-direction and does not mean that heterogeneous SVs could not be used for mirroring.

When an ASAD could not access a copy of data, the I/O is directed to one of the alternates that is most closely located to the operating site. Every copy of a given data is updated simultaneously, when possible. During writes, a block is considered to be written when all the copies have been written to available sites.

6.3 Creating an ASAD

The first thing in a series of activities involved with an ASAD is its creation. Creating an ASAD entails the process of defining and integrating all the SVs that constitute its ASA in order to produce the logical space. The creation of an ASAD is achieved by a command whose parameters define the necessary attributes. The attributes include the SVs, ASAD unique name, current number of SVs, operating mode, etc. A unique identification tag is generated for each ASAD during its creation. The operating mode specifies whether the ASAD allows read-only, read-write or write-only operations. The list of SVs include the mirrors and each SV is specified with its *host machine address*, *major* and *minor* numbers. A header is built for each ASAD which comprises the aforementioned pieces of information, a *magic string*, and other pertinent hardware characteristics of the storage devices in the ASA. The local machine that creates an ASAD sends a message to the host machine of each of the SVs requesting for its size, bandwidth, sector size, etc. A copy of an ASAD header is stored at a specified location in every SV in the ASA at the request of the site creating an ASAD. The header must be successfully written to the ASA and the associated mirrors before an ASAD can be said to have been successfully created. The different sections of an SV (as described below) are also created at this time. After an ASAD has been successfully created, a stanza is created for it in the configuration file where all the necessary attribute-value pairs are enumerated.

6.3.1 Storage Volume Layout

An SV consists of physically contiguous string of sectors. The sectors are sequentially number form zero. Each SV is partitioned into 3 sections, namely, **Reserved, Header**, and **Data** sections as shown in Figure 6. The Reserved section consists of the first specified number of sectors which are reserved for special system data such as the boot image. The GDMS does not tamper with this part of an SV. The Header section contains all the header data about the ASAD to which an SV belongs. The Header is usually built from a majority of the information specified during the creation. A duplicate of the header information is kept in another part of the SV. The size of the Header section is staticly defined from the maximum size of an ASA. The Data section contains the blocks where the data objects are actually stored. It occupies the rest of the SV space.

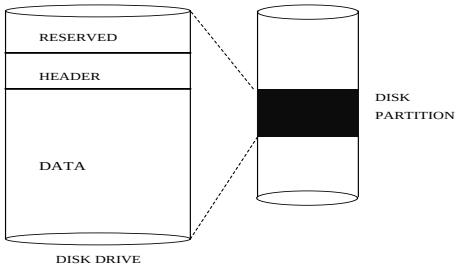

Figure 6: An SV's sections.

6.4 ASAD Configuration Process

An ASAD creation produces and stores the necessary information for controlling an ASA. However, all the information produced are stored in the specified sections of an SV. The configuration process entails the generation of the necessary kernel data structures needed to access an ASAD. The configuration file is

read for a specified object and the associated SVs are validated. The validation involves the comparative analysis of each SVs header information. The configuration of an object fails if there exists an inconsistency between the header information of the SVs. The configuration process also makes sure that all the SVs of an ASAD are available, i.e., on-line.

Consistent with UNIX-based operating systems, an ASAD is associated with a major and minor number for the *device switch table*. The device switch table entry corresponding to an ASAD major number contains the function addresses for the *strategy, read, write, open, close, ioctl*, etc., routines for the GDMS. The minor number is used to store the private information, i.e., header information, about a specific ASAD. Each site is responsible for controlling the accesses to its storage devices that belong to a given ASA. We configure an ASAD at every site that contains one or more of its SVs. The configuration of an ASAD at more than one site is accomplished by sending appropriate instruction messages to the sites with the pertinent configuration information. Every site spanned by an ASAD's ASA is a server for the ASAD. It is an uncompromising requirement that an ASAD must be configurable at all the applicable sites for a configuration process to be successful. At the conclusion of the configuration of an ASAD, a special device file is created whose major number is the major number of the GDMS system and minor number is the ASAD minor number. The name of the special device file is one of the attribute-value pairs that is specified during an ASAD creation.

6.5 Making a File System on an ASAD

Being a pseudo-device driver, the GDMS is designed to run beneath any UNIX-based file system or any subsystem that requires secondary storage services. Figure 7 shows the system setup when the GDMS is available. It shows that different system components that require secondary storage interface with the GDMS instead of the device drivers. Acting as a pseudo-device driver, the GDMS interfaces with the real device drivers of the storage devices that it manages. The GDMS can manage a heterogeneous configuration of storage devices. A file system is created on an ASAD by using the file system's normal command and appropriate parameters. The special device file name used is the one created after the ASAD configuration. The existence of the GDMS may be transparent to the file system or the file system may be enhanced to exploit some of its characteristics.

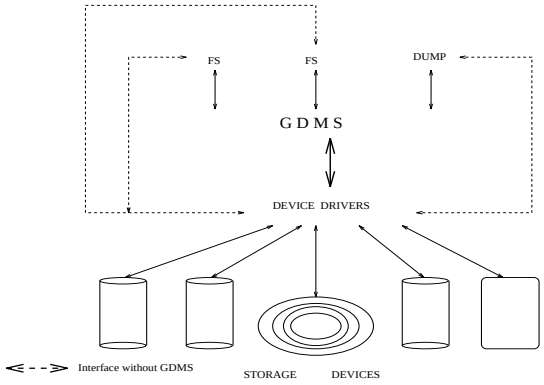

Figure 7: System setup with and without GDMS.

6.6 Extension and Reduction of an ASAD

It is possible to extend or reduce an existing ASAD that already contains some data. An ASAD is extended by adding one or more SVs to its ASA. A specific command interface is used to extend an ASAD. The command parameters include the name of the ASAD to extend and the SVs with associated mirrors, if applicable. A copy of the header is obtained from one of the SVs of the target ASAD and subsequently updated with the new SVs. The new header is used to replace the old headers in the applicable SVs and a new configuration file attribute-value pairs are generated for the ASAD to replace the old ones after a successful extension. The extended space in an ASAD becomes active after the next configuration of the ASAD. A mechanism is provided by the DMDMS for utilizing the extended space.

An ASAD is reduced by removing one or more SVs from its ASA. Like extension, the ASAD must be offline and a command interface is used similarly. In the event that some valuable data may be lost as a result of reduction, the user is advised accordingly and the discretion to proceed or terminate is left to the user. After a successful reduction, the configuration file is updated for the ASAD. After a reduction, the removed SVs become unavailable after the next configuration.

6.7 Data Access and Manipulation

One of the most critical issues in a computing environment like the GDMS is the control of data access and manipulation in order to maintain consistency in the system. We have previously identified some of the important issues and techniques that may be utilized to control data accesses in a distributed environment. In this section we discuss how those problems are re-

solved in the GDMS. Controlling accesses and manipulations of some data are more reliable when every operation to the data has to go through a single channel. The system components that interface with the GDMS request for data retrieval or update by specifying a logical address within the ASAD. The requesting site determines the specific storage device or storage devices spanned by the I/O request and also the locations of the devices. In the case of reads, the client site (local site) may use network or other pieces of information to determine the closest site to the local site for any mirrors of the target data. Instruction messages for I/O operations are generated and sent to the applicable hosts. The process just described is the *localized service* approach discussed in Section 4.2. Each site maintains in-core copies of the most recently accessed blocks of the storage devices that are attached to it. The specific cache replacement strategy used is implementation and environment dependent. Every necessary update to any data is immediately directed to the target site and subsequently propagated to permanent storage. The buffers are periodically flushed to storage devices by each site. To put it laconically, all the control mechanisms for any SV or data are locally generated and maintained. These activities support the *write-through* update policy and the UNIX sharing semantics.

6.8 Dichotomy of Cache Update Policies

In order to limit the frequency of I/O requests to the secondary storage devices, file systems maintain buffer caches at the file system level. The cache update policy employed by a file system or a data management system is system dependent. Since a file system or GDMS client and the GDMS maintain different cache levels, the difference in the caching schemes employed can greatly affect the data consistency in the system. The GDMS utilizes *write-through* policy. The caching mechanisms employed by the GDMS and its client are transparent to each other. For example, if a GDMS client uses a non-*write-through* policy, then after an update to an object, the changes may not be propagated to the master copy. Therefore, the server for the master copy is oblivious of the changes. Subsequent accesses to the object will then obtain stale data, thereby, violating the expected observance of the UNIX sharing semantics. In view of the existence of multiple levels of caching between the GDMS and its clients, and the necessity for data consistency, it is necessary that the GDMS and its clients employ the same update policies. In that way, changes are immediately propagated to the master copies for data consistency.

6.9 Configuration Command and Daemon

Every site spanned by an ASA has an access to a configuration command. The configuration command dictates what action to perform on an ASAD. The action must be either to *create, configure, extend, reduce*, or *de-configure* an ASAD as described earlier. The control of the execution of those activities is handled by a configuration daemon. The configuration daemon can be running in every applicable site or be designated as a centralized program located in a well known site. The configuration command communicates with the daemon via a known *pipe* to specify which action is needed to be performed by the daemon. A pipe is a UNIX special file that is used for communication between two or more processes. Figure 8 shows a schematic representation of some the interactions between the configuration command, configuration daemon, and the GDMS servers during the creation of an ASAD. Figure 9 shows the interactions between the configuration command, configuration daemon, and the GDMS servers during an ASAD configuration. The communications between the configuration daemon and a GDMS server, and all inter-server communications are via message passing.

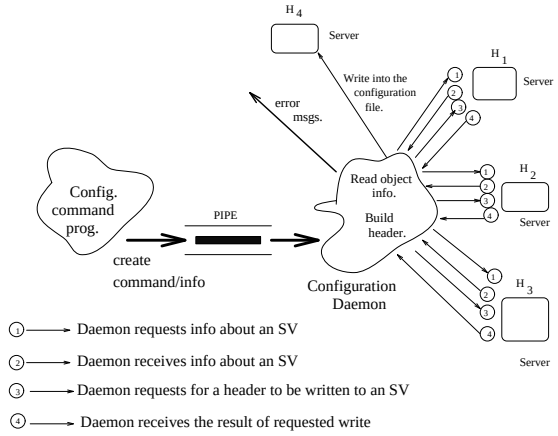

Figure 8: Operations when creating an ASAD.

The de-configuration of an ASAD entails the destruction of all the in-core information maintained at each applicable site about the target ASAD. A de-configured ASAD becomes inaccessible. Before an ASAD is de-configured, all pertinent information are flushed to permanent storage, all open files must have been closed, and the appropriate FS unmounted. The closing and unmounting of an ASAD helps in completing pending writes.

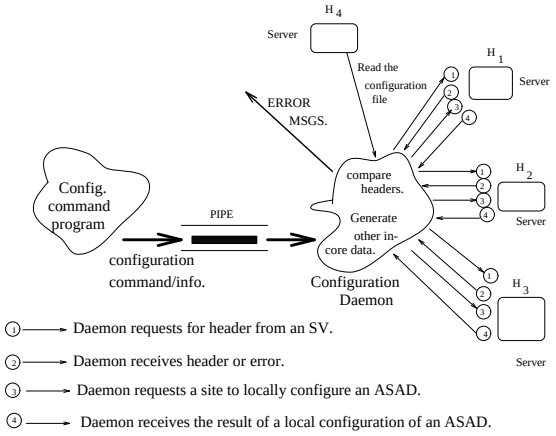

Figure 9: Operations when configuring an ASAD.

6.10 GDMS Local Server Processes

Each site that contains a storage device of an ASAD runs a server process. The server process is responsible for controlling all accesses and manipulations of the data stored in its local SV. Each server process, depending on an operation, may act as a client to another server process.

A local GDMS process is implemented with *lightweight* processes. Each server process generates a number of lightweight processes where each lightweight process is responsible for servicing a single request.

7 Conclusions and Contributions

In this paper, we have presented a global data management strategy for a logical storage space based on storage devices that are attached to computing systems that may be locally or remotely interconnected via a network. We discussed the functional characteristics and capabilities of the strategy vis-à-vis the current techniques. The processes for the creation, configuration, making file systems, accessing data, extension, and reduction of an ASAD are described.

The strategy presented incorporates the vital issues in distributed data management that are necessary for data access, consistency, availability, and reliability. It presents all the management interfaces and processes necessary for creating, configuring, and managing the logical storage spaces defined across storage devices attached to both local and remote machines. It also maintains the element of transparency which is a *sine qua non* for distributed systems. The strategy provides the framework for utilizing logical space management for diverse configurations of storage devices for different file systems.

References

[1] **G. Popek, B. Walker**, (Eds) The LOCUS Distributed System Architecture, *MIT Press*, Cambridge, Mass., 1985.

[2] **R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyone**, Design and implementation of the sun network file system, *Proc. USENIX 1985 Summer Conf.*, June, 1985, pp. 119-130.

[3] **M. Nelson, B. Welch, J. K. Ousterhout**, Caching in the Sprite network file system, *ACM Trans. Comput. Syst.*, Vol. 6, No. 1, Feb. 1988.

[4] **J. K. Ousterhout, A. R. Cherenson, F. Douglis, M. N. Nelson, B. B. Welch**, The Sprite network operating system, *IEEE Comput.*, Vol. 21, No. 2, Feb. 1988, pp. 23-36.

[5] **J. H. Morris, M. Satyanarayanan, M. H. Conner, J. H. Howard, D. S. H. Rosenthal, F. D. Smith**, Andrew: A distributed personal computing environment, *Commun. ACM*, Vol. 29, No. 3, March 1986, pp. 184-201.

[6] **P. Chen, D. Patterson**, Maximizing Performance in a Striped Disk Array, *Proc. 1990 ACM SIGARCH 17th Intern. Symp. on Comp. Arch., Seattle, WA, May 1990*, pp. 322-331.

[7] **M. Y. Kim**, Synchronized Disk Interleaving, *IEEE Trans. on Computers*, Vol. C-35, Vol. 11, November 1986, pp. 978-988.

[8] **P. V. Rangan, T. Kaeppner, H. M. Vin**, Techniques for Efficient Storage of Digital Video and Audio, *International Workshop on Multimedia*, Feb., 1992, pp. 68-85.

[9] **W. Richard Stevens**, UNIX Network Programming, *Prentice Hall*, Englewood Cliffs, New Jersey.

[10] **LVM Design Team**, Logical Volume Design, *Open Software Foundation*, 1990.

[11] **D. Patterson, G. Gibson, R. Katz**, A case for Redundant Arrays of Inexpensive Disks(RAID), *In Proc., 1988 ACM SIGMOD Int'l Conf.*, May 1988, pp. 109-116.