Sept. 2006, Vol.21, No.5, pp.648–664 *J. Comput. Sci. & Technol.*

Massive Storage Systems

Dan Feng (冯　丹) and Hai Jin (金　海)

School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, P.R. China

E-mail: dfeng@hust.edu.cn; hjin@hust.edu.cn

Received April 2, 2006; revised July 24, 2006.

Abstract To accommodate the explosively increasing amount of data in many areas such as scientific computing and e-Business, physical storage devices and control components have been separated from traditional computing systems to become a scalable, intelligent storage subsystem that, when appropriately designed, should provide transparent storage interface, effective data allocation, flexible and efficient storage management, and other impressive features. The design goals and desirable features of such a storage subsystem include high performance, high scalability, high availability, high reliability and high security. Extensive research has been conducted in this field by researchers all over the world, yet many issues still remain open and challenging. This paper studies five different online massive storage systems and one offline storage system that we have developed with the research grant support from China. The storage pool with multiple network-attached RAIDs avoids expensive store-and-forward data copying between the server and storage system, improving data transfer rate by a factor of 2–3 over a traditional disk array. Two types of high performance distributed storage systems for local-area network storage are introduced in the paper. One of them is the *Virtual Interface Storage Architecture* (VISA) where VI as a communication protocol replaces the TCP/IP protocol in the system. VISA's performance is shown to achieve better than that of IP SAN by designing and implementing the vSCSI (VI-attached SCSI) protocol to support SCSI commands in the VI network. The other is a fault-tolerant parallel virtual file system that is designed and implemented to provide high I/O performance and high reliability. A global distributed storage system for wide-area network storage is discussed in detail in the paper, where a Storage Service Provider is added to provide storage service and plays the role of user agent for the storage system. Object based Storage Systems not only store data but also adopt the attributes and methods of objects that encapsulate the data. The adaptive policy triggering mechanism (APTM), which borrows proven machine learning techniques to improve the scalability of object storage systems, is the embodiment of the idea about smart storage device and facilitates the self-management of massive storage systems. A typical offline massive storage system is used to backup data or store documents, for which the tape virtualization technology is discussed. Finally, a domain-based storage management framework for different types of storage systems is presented in the paper.

Keywords massive storage, SAN, NAS, intelligent storage, storage virtualization, data grid

1 Introduction

According to IBM and InfoWeek, the amount of stored information is growing at a rate of 700 percent per year, mainly driven by e-Business, the digital revolution, and the explosive growth of Internet, as well as enterprise data.

The storage technology has seen rapid advances over the past fifty years since introduction of the IBM 305 RAMAC in 1956. Aside from the increases in disk drive capacity and speed, enterprise disk and tape technology have morphed into a network resource through *Network-Attached Storage* (NAS) and *Storage Area Network* (SAN). In conjunction with networking, storage technology has also increased its brainpower with the addition of cheap and powerful microprocessors. These developments result in a new generation of intelligent storage networks that can automate tasks, maximize utilization, and lower operating costs.

1.1 Direct-Attached Storage

Direct-attached storage (DAS) is defined as a collection of storage disks attached to a single server via a cable. DAS attributes its roots to the era of computing before the networking revolution, an era in which infrastructure design was not concerned with the future needs of data sharing or information management but focused on applications and servers, not data and storage. Over time, these deployments revealed the limitations of DAS, such as single server access, labor-intensive management, poor scalability, costly upgrades, low utilization and multiple "single points of failure".

To overcome the shortcomings of DAS, network has been adopted to construct storage systems. This infrastructure design offers a way to share storage resources using one of the two technologies, namely, *Network Attached Storage* (NAS) and *Storage Area Network* (SAN).

1.2 NAS

The NAS system has several advantages. For instance, it is relatively easy to use, and it leverages the standard Ethernet infrastructure to connect clients to the storage.

RAID[1] (*Redundant Array of Inexpensive Disks*), proposed by David A. Patterson *et al.* in 1988, led to high performance and reliable storage systems. As the

Survey

This paper is supported by the National Natural Science Foundation of China under Grants No.60125208, No.60273074, No.60303032, No.69973017, and the National Grand Fundamental Research 973 Program of China under Grants No.2004CB318201, No. 2003CB317003.

most common device of data storage in DAS, RAID is a category of disk drives that employ two or more drives in combination for fault tolerance and performance. RAID is used frequently on servers but is not generally necessary for personal computer. There are a number of different RAID levels: 0/1/2/3/4/5/6/0+1/10/7/S.

NAS systems are characterized as specialized servers with optimized file systems and thin or stripped-down operating systems that are tuned for the requirements of file serving. They communicate with clients using the NFS (*Network File System*) protocol, the CIFS (*Common Internet File System*) protocol, or both.

However, the NAS system suffers from the same scalability problems associated with DAS. The filer protocols are often incompatible with certain common applications.

1.3 SAN

SAN offers a block-level protocol (e.g., SCSI) for clients to access data over the network. This block-level protocol is embedded into a network protocol such as *Fibre-Channel* (FC) and *Internet SCSI* (iSCSI). Unlike the network file protocols (e.g., NFS and CIFS) that NAS uses, these block-level protocols provide complete application compatibility, plus performance results that match and often exceed those of DAS.

However, despite its benefits, the number of actual FC deployments remains relatively modest. This is due to several new problems created by FC, and once again several key problems left unresolved. For instance, it is very expensive to deploy and grow FC-based SANs — prohibitively so for most organizations.

Unlike the FC technology, utilizing an ordinary IP network, the iSCSI protocol transports block-level data between an iSCSI initiator on a server and an iSCSI target on a storage device. With the adoption of the iSCSI standard by the *Internet Engineering Task Force* (IETF) and strong vendor support from industry leaders such as Microsoft, Cisco and Network Appliance, iSCSI has rapidly matured into a viable, alternative SAN technology.

1.4 Object-Based Storage

There are increasing demands for storage capacity, throughput and ease of management. For example, it is believed that the quality expectation drives technology[2], with the highest level of quality being demanded by the Tri-labs and the oil and gas industry, requiring throughput of 1GB/s and 1.2GB/s, respectively. For technology to improve and bridge the chasm, the need for such performance must be demanded by the masses. This includes the need for both file and file aggregate bandwidth improvements. Bigger file server boxes[2] cannot keep up with the growth of demand or supply. The Lustre File System[3] and the Panasas ActiveScale File System[4] are based on the *object-based storage* (OBS)[5] technology, having been designed from the ground up to meet the demands of the world's largest high-performance cluster systems.

The contributions of OBS can be regarded as the convergence of NAS and SAN. Users and applications get file interface from NAS and block interface from SAN, respectively. Like files or blocks, objects are primitive, logical units of storage that can be directly accessed on a storage device.

By proposing a "multi-layer scalable storage object" concept, we want to unify and update the network storage, and to construct an object-based petabyte-scale storage system. The main idea of "multi-layer scalable storage object" is to move the parts of program that are used to access the disks in host system into the storage system, to encapsulate them into objects containing data and operations, and to endow the storage objects with the characteristics of intelligence and the ability of initiative service.

1.5 Massive Distributed Storage Systems

There have been many research prototypes, middleware, or protocols of distributed storage systems or global storage systems, such as DPSS[6], SRB[7], OceanStore[8], and GridFTP[9].

Distributed Parallel Storage System (DPSS)[6] is a scalable, high-performance, distributed parallel data storage system, which provides a high performance data handling architecture for building high-performance storage systems from low-cost commodity hardware components.

GridFTP[9] is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP. It provides a universal grid data transfer and access protocol for secure, efficient data movement in grid environments. It extends the standard FTP protocol, and provides a superset of the features offered by the various grid storage systems currently in use.

Storage Resource Broker (SRB) of the San Diego Supercomputing Center (SDSC)[7] is a client-server middleware that provides a uniform interface for heterogeneous data resources over a network and accesses to replicated data sets. In conjunction with the Metadata Catalog, SRB provides a way to access data sets and resources based on their attributes rather than their names or physical locations.

OceanStore[8] is a global persistent data store designed to scale to billions of users by using a peer-to-peer infrastructure. It provides a consistent, high-available, and durable storage utility atop an infrastructure comprised of untrusted servers. It employs a Byzantine-fault tolerant commit protocol to provide strong consistency across replicas.

1.6 Other Trends

Recent efforts on storage systems research are stepping toward an intelligent storage system. More storage

devices embedded with powerful ASIC designs are capable of considerable processing. Some research efforts on intelligent storage design have explored two promising enhancements: impressive storage interface and embedded computational power. Active Disk[10], self-star[11], IBM's Autonomic Computing[12], EMC's AutoIS[13] and Semantically-Smart Storage[14] all focus on the intelligent storage design.

Future storage technologies will be built on previous research and developments in access, scalability, interoperability, and long term stewardship of globally distributed storage. For example, recently, peer-to-peer (P2P) and Grid are both concerned with the pooling and coordinated use of resources within distributed communities and have emerged as a popular way to share huge amounts of data. P2P systems decentralize the management of massive storage systems and harness unused resources on computers distributed across the world. Unlike traditional client/server systems that essentially depend on many servers with high reliability and availability, in P2P systems, failure of a few nodes may not result in a sharp decrease in data availability because all nodes are peers and the remaining peers may provide the required data. Both approaches have seen rapid evolution, widespread deployment, and successful application. Other key trends include storage system security, management, storage compliance with regulation, fixed content optimizations, tiered storage, storage virtualization, policy management[15] and storage management software.

In China, extensive research efforts in the field of massive storage system have been supported by the 973 Program, the 863 Program and the National Natural Science Foundation of China, aiming at addressing such problems prevalent in the current server systems as I/O bottleneck, storage capacity, data robustness and so on. Significant results have been achieved from these efforts.

In this paper, we will present some of these achievements from several storage systems under development: a storage pool with network-attached RAID, an object storage system, a high performance distributed storage system for local area network storage, and a global distributed storage system for wide area network storage. All of these storage systems are online massive storage systems. A typical offline massive storage system are used to backup data or store documents, for which the tape virtualization technology is described in the paper also. By investigating the storage system architecture, storage system controller software and the management software, among other things, different storage systems have been developed to achieve high availability, high reliability, high performance, high scalability, and high security.

The rest of the paper is organized as follows. A storage system of Network-Attached RAIDs with Heterogeneous Channels is presented in Section 2. A Petabyte-Scale Object Storage System and its key technologies are presented in Section 3 to show the trend of incorporating more intelligence into storage systems. In Section 4 two types of high-performance distributed storage systems for local-area network storage are described, i.e., the *Virtual Interface Storage Architecture* (VISA) and a fault tolerant parallel virtual file system. A global distributed storage system for wide-area network storage is discussed in detail in Section 5. The tape virtualization technology for offline massive storage systems is described in Section 6. Section 7 presents a domain-based storage management framework for the heterogeneous WAN-based environment. We conclude the paper in Section 8.

2 Storage Systems of Network-Attached RAIDs with Heterogeneous Channels

2.1 Network-Attached RAIDs Systems with Heterogeneous Channels

In massive storage systems, disk arrays are mostly used for capacity and performance reasons. Disk arrays can make multiple disk drives working in parallel to improve both the throughout and the I/O rate. Normally a disk array is connected to the server only by a peripheral channel. This kind of connection organization confines the data-moving path between storage devices and network to pass through the server, thus confining the advantages brought by storage devices.

Despite of the rapid development of network technology and high-performance storage device technology, and the continued increase in processing power, the server remains a performance bottleneck by sitting on the data-moving critical path and spending too much time and power in controlling I/O operations to move data multiple times inside the system. To relieve the server of the burden and increase storage performance, the functions of storage device are promoted by adding network protocol supports, thus delivering the retrieved data directly to the network by storage devices without going through the system memory and the buffers of network interfaces.

To this end, a different architecture for massive storage system, called the Heter-RAID (*Heterogeneous channel RAID*) system, shown in Fig.1, is proposed and designed for better delivering of large amount of data over the network. Heter-RAID consists of file server and Heter-RAID device. The Heter-RAID device is a key device of the system. It is a disk array with additional intelligence and functions that support the network protocol stack and dynamic I/O strategy adjustment. Apart from the usual components of a disk array, the Heter-RAID controller has a network interface to link the disk array directly onto the network. In this way, it gets dual channels: the traditional peripheral channel and the network logic data channel. The relationship between the server and the storage subsystem is no longer only a simple master-slave relationship, but now also a peering-entry relationship in the network. The relationship between the storage subsystem and the client also changes from one of invisible device screened by the server to one of

visible peer node on the network.

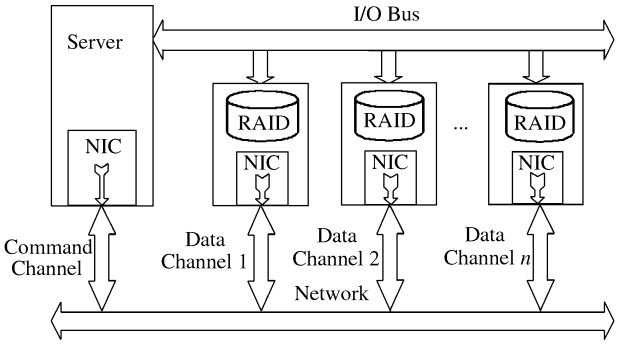

Fig.1. Block diagram of the Heter-RAID workflow.

In Heter-RAID, I/O tasks from server or clients accepted by command implementer enter the command queue according to their arrival times. Then the queue optimizer schedules the tasks in the queue at the user request level, and decomposes and rearranges them to get the optimized I/O command queue.

Scheduling is determined by a given strategy. I/O implementer schedules and operates the I/O commands at the I/O level. The resultant data is transferred by the data redirector either to the server through the peripheral channel or to clients through the network interface.

The architecture avoids expensive store-and-forward data copying between the server and Heter-RAIDs when clients download/upload data from/to the server. For example, when a client requires data from the server, the read command is sent to the disk array through the peripheral bus by the server, but the data is directly transferred from the disk array to the client. So the latency is lower than that with traditional architecture. The system performance of the proposed architecture is evaluated through a prototype implementation based on the logical separation in the File Transfer Protocol. In a multi-user environment, its data transfer rate is 2–3 times higher than that with a traditional disk array, and service time is about 3 times shorter.

2.2 Multi-Task Pipelining I/O Scheduling in Network Storage Systems

Task parallelism and pipelining are two common forms of concurrency. Task parallelism assigns different tasks to different concurrent objects; whereas, task pipelining divides a repetitive task into specialized stages so that successive tasks can be overlapped in time when they are executed in distinct stages simultaneously. Network storage systems must meet two requirements to benefit from the pipelining technology: the system can get information of the next command before the completion of the current command, and different units (stages) can operate simultaneously without resource contention[16].

I/O path[17] involves every phase of information processing, transmission and storage. Shortening an I/O path helps decrease the protocol layer, reduce complexity, shorten overhead or lower resource consumption. In modern network storage systems, more complex storage protocols and network protocols translate to more difficult management. Moreover, additional overhead introduced by an unsuitable I/O path design reduces the efficiency of network storage. I/O scheduling implemented by pipelining schedule can overlap part of the overhead in an I/O path and achieve high storage performance[18].

Fig.2 shows a multi-task pipelining I/O scheduling timing diagram. In the same pipelining stage, the number of sub-tasks is called the degree of pipelining parallelism, which generates synchronization overhead. A large workload can be segmented into a number of smaller sub-workloads at a stage in the I/O path, so as to process these sub-workloads in parallel at the same stage.

In Heter-RAID[19], virtual SCSI commands encapsulated with operation type (read/write), start sector, sector number and other information are adopted to execute I/O operation. There are multiple virtual SCSI commands in the Heter-RAID command queue within system resources. Pipelining producer/consumer policy divides the I/O cycle of virtual SCSI command into different stages and uses a buffer technique to smooth work speed of different function components, which overlaps disk I/O and CPU computation to improve system performance.

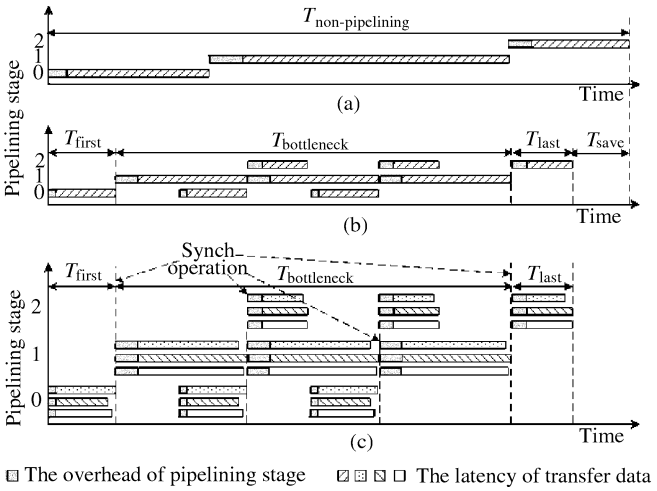

Fig.2. Multi-task pipelining I/O scheduling timing diagram. (a) Parallel degree is 1 for three-stage non-pipelining. (b) Parallel degree is 1 for three-stage pipelining. (c) Parallel degree is 3 for three-stage pipelining.

According to the overlapping degree of I/O scheduling processes, pipeline operations of a network attached storage system can be divided into two methods, namely, the fixed pipeline scheduling, and the flexible pipeline scheduling[16]. Multiple processes will be executed by a fixed scheduling sequence in the fixed pipeline scheduling. Otherwise, the flexible pipeline scheduling examines the completing sequence and overlaps multiple processes freely. The Heter-RAID scheme has doubled the per-

formance of average I/O response time and throughput (improved from 78MB/s to 147MB/s).

3 Petabyte-Scale Object Storage Systems

Hints are sometimes necessary to facilitate computer system design, and it is important to use hints to speed up normal execution[20]. In HOSS (*HUST object storage system*), objects can give useful hints to the storage system, in the form of object attributes (e.g., size, access, resilience, security, semantics) and object methods, thus making the storage system effectively smarter. Hints about an object's access pattern, reliability, availability, accessibility, and lifespan can aid in a variety of ways including improving the object's layout on an object storage node, QoS[21] and increasing the effectiveness of management and data caching.

3.1 HUST Object Storage System (HOSS)

HOSS comprises at least a scalable *Object-based Storage Controller* (OSC)[22], which combines the advantages of the object-based interface and the embedded processing power. An OSC has a flexible way to describe object attributes, so an OSC can store any kind of object and is adaptive to a wide range of applications. Furthermore, users can offload operations (or methods) in an OSC. An object is treated as a stream. User can associate a chain of methods[23] with the stream, and an OSC can automatically execute a series of method taking the stream as input.

An OSC is built from off-the-shelf components, as shown in Fig.3. Like other intelligent disks, it has processor, memory, network interface and block-based disk interface, and has the ability to perform intelligent processing on stored objects.

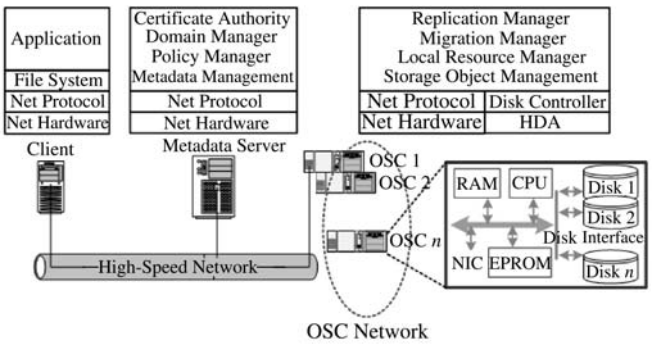

Fig.3. OSS architecture.

In HOSS, *Metadata Server* (MS) provides the information (global object metadata) necessary to directly access objects, along with other information of the data including its extended attributes, security keys, and permissions (authentication). At the same time, MS provides objects layout information in OSCs. For example, MS stores higher-level information about the object, such as object ID (OID), object name, object type (e.g., file, device, database table) and storage map (e.g., ⟨OSC1, partion1, OID1, OSC2, partion2, OID2, stripe unit size of 64KB⟩), which are not directly interpreted by OSC. OSCs expose an object-based interface, and the access/storage unit is object, like a file, a database table, a medical image or a video stream. OSCs manage local object metadata (data organization in disks). Clients contact MS and get the information about objects. OSCs receive and process those requests with some predefined policies.

3.2 Scalable Object Attributes and Methods

An object can be extended to define an object in HOSS. An object is variable-length and can be used to store any type of data. Thus different types of data has different attributes. If we use a uniform format to represent attributes for all types of objects, more disk space may be wasted while saving objects with their attributes in disks. We propose a scalable way to present attributes[23], called attribute card (AC). HOSS can support a new type of object by simply defining a new AC for it. Such a scalable attribute representation makes OSC suitable for any application.

Object attributes can tell the storage system how to deal with an object, for instance, the level of RAID to apply, the size of the capacity quota or the performance parameters required for that data. Once OSC accesses to attributes describing the objects, it can use these attributes to achieve improvements at the device level. For example, OSC might make sure that the hot spot object is available more quickly, or that previous versions of objects are automatically saved when it detects that objects are being changed. When OSC understands the meaning of data it contains, such as backup or QoS requirements, object access patterns (sequential or random), or relationships to other objects, it can use that information to create new storage applications that take advantage of OSC's ample processing power,.

In the traditional way, a block-based disk can only perform READ and WRITE operations on data. An intelligent device changes this by allowing users to upload application-specific operations at the disk level and perform operations on data when receiving requests from users. Furthermore, an intelligent device must provide more flexible and scalable operations that can be applied to more application fields. An OSC provides good scalability by supporting the Method Chain[23].

Clients can associate an object with specific methods in two ways that are transient and persistent. In the transient way, methods can be built when an object is opened and destroyed when the object is closed. In the persistent way, methods for an object can be created/destroyed through a register/un-register request to HOSS. After an object is associated with its methods through registration, the operations in the Method Chain are always implicitly performed.

3.3 Adaptive Policy Triggering Mechanism

Besides the above scalability of HOSS, the extension of the storage system is confronted with problems of management. With scalable object attributes and methods, an abundance of clues (or hints) are obtained to guide or direct the solution of self-management by HOSS. Therefore, HOSS is more facilitating in setting up mass storage systems than either NAS (file-level interface) or SAN (block-level interface).

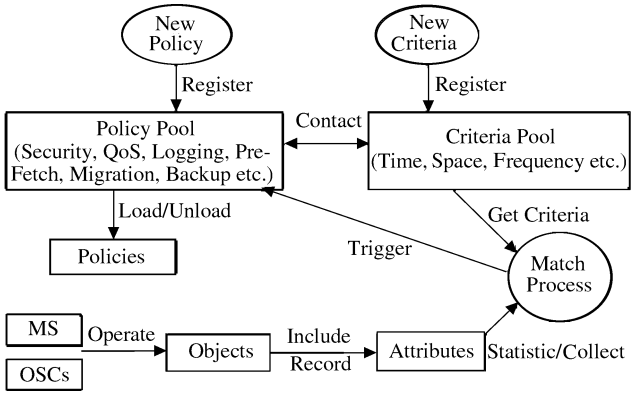

Fig.4. Adaptive policy triggering mechanism.

A "policy" can be thought of as a coherent set of rules to administer, manage, and control access to network resources. Fig.4 shows the design of an *adaptive policy triggering mechanism* (APTM). In Fig.4, criteria and policies are separated because one policy may correspond to several criteria while one criterion may adapt to different policies. The criteria pool is filled with general values regarded to be useful by one or more of the management policies. Most popular criteria such as time, frequency of access, capacity and size are initially registered to the pool. A new policy registered into the policy pool has to contact the corresponding criteria in case they are not already registered. So HOSS can provide a solution for a dynamic loading or unloading policy.

APTM ensures that the registered criteria are updated on the performance of storage system state. These policies themselves are just descriptions of how to implement system management functions and specify system states and how to response to them. These may include any proposed storage system management policy. When clients or the system operates on objects, HOSS records these information by object attribute values. For the adaptive policy triggering, those correlative policies are triggered and therefore have the largest effect on the storage system performance. Additionally, the adaptive policy triggers the policy depending on the matching process between object attribute values and criteria from the criteria pool.

3.4 Object Assignment Algorithm

By distributing objects across many devices, object storage systems have the potential to provide high throughput, reliability, availability and scalability[24]. AFS[25], Lustre[3], Panasas[4], GFS[26], Coda[27] and GPFS[28] have investigated hierarchy management of distributed file systems, but relatively little research has been aimed at improving the efficiency of object allocation in large scale object-based storage systems.

HOSS represents files as sets of objects stored on self-managed object storage nodes (composed of OSC and disks). The algorithm used for object allocation determines the performance of the system at the beginning of the communication process. It affects the workload among the devices, and it also influences the OSC-level parallelism.

```
INITIALIZATION:
    Get number of devices (the total number of OSCs) and
    set dev[] empty.
Objects_allocation (the size of file)
Input: The size of file
1:   if the size of file < 512KB then //small file
2:       Hashing();
3:   else                                          //large file
4:       if number of devices < 20 then
5:          N = number of devices;
6:       else if 20 <= number of devices <= 40 then
7:          N = 20;
8:       else
9:          N = 40;
10:      endif
11:      Sort dev[1], dev[2], ..., dev[number of devices] by
            performance;
12:      if the size of file <= N * 512KB then
13:         Fragment_striping (dev[1], dev[2], ...,
               dev[the size of file/512KB]);
14:      else
15:         Fragment_striping (dev[1], dev[2], ..., dev[N]);
16:      endif
17: endif
Devices parameters in Sort dev[1], dev[2], ..., dev[number
    of devices] by performance;
typedef struct dev_info {
    int          type;          //OSC types
                                      (RAID, JBOD, single disk)
    int          busy;          //busy status
    uint64_t     freesp;        //free capacity
    uint64_t     partitions;    //partitions in the device
    ulong        ip;            //IP address
    struct dev_info*next  ;
}DEVINFO;
```

Fig.5. Static object allocation algorithm.

In HOSS, a static data layout algorithm considers the following three questions. The first is how to draw the boundary between small and large files. The second is how to determine the optimal number of objects mapped from one file. The third is how to select OSCs for parallel transmission. [29] integrates two techniques that are hashing and fragment-mapping based. The algorithm is shown in Fig.5. When the file is small, it is converted to a single object and directly mapped to an OSC by hashing. If the file is large, it is converted to multiple objects and each object will be distributed to some, possibly different, OSC.

4 High Performance Distributed Storage Systems

The high performance-cost ratio of cluster computing has made it the most popular platform for high-performance computing today. Nevertheless, as with the traditional massively parallel computers, I/O remains a challenge. For example, scientific applications running on clusters, such as astrophysics simulation and global climate modeling, usually require the input and output of large amounts of data. Therefore, I/O performance is crucial and tends to be the bottleneck for the overall system as well as application performance.

Also, storage industry wakes up to these actualities. In MSST 2005, the "top100io" storage systems, sponsored by the IEEE Computer Society Mass Storage Systems Technical Committee (MSSTC), was originally intended to track and detect trends in high performance storage systems.

4.1 Virtual Interface Storage Architecture

VISA (*Virtual Interface Storage Architecture*)[30] is a distributed storage system based on *Virtual Interface* (VI), where VI as a communication protocol replaces the TCP/IP protocol in the system. In order to implement remote accesses in the storage system, a front-end server running the iSCSI protocol is added to the system.

Although VI was originally designed primarily for message transfers in a cluster environment, it is also suitable for distributed storage systems and SANs because of its high-bandwidth and low-latency. A new network storage system, called VISA, is designed based on the VI communication protocol. The architecture is shown in Fig.6.

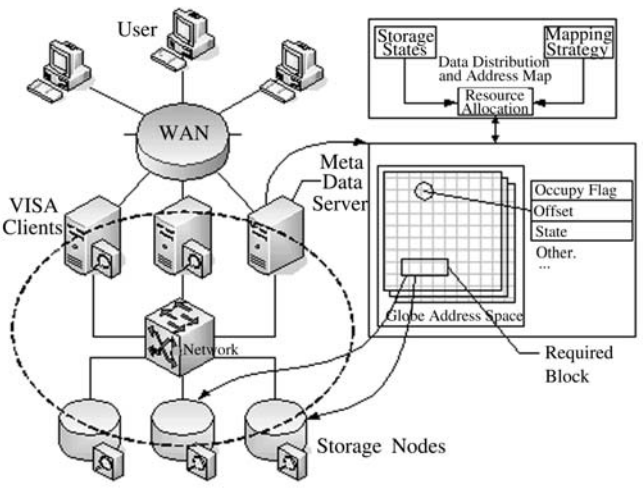

Fig.6. Virtual interface storage architecture (VISA).

VISA contains three main types of components, which are one or more metadata servers, a number of storage nodes that provide storage resource and a number of client nodes that consume the storage resource. All the three types of components are connected by a VI network.

The metadata server is a pivotal part in VISA. First, it holds physical storage information of all storage nodes and keeps the information up-to-date. Second, it has a function of virtualizing the storage: the client nodes can access the storage resource transparently without client's knowledge of physical data layout. Finally, it can actively send messages to storage nodes to inquire the status of all storage nodes. It can also passively accept messages informing storage resource changes sent by storage nodes, thus dynamically detecting node arrivals at or departures from VISA. This feature allows VISA to be expandable. The VISA design can accommodate two or more metadata servers to ensure the scalability and availability of the system.

Storage nodes are another important part in VISA. Storage nodes not only provide storage resource for clients but also ensure the reliability of clients' data. By using RAID and some other technologies, it can achieve good reliability and high performance. Cooperating with the metadata server(s), it can respond to the status inquiries sent by the metadata server(s) and actively send the information to metadata server(s) when the storage resources change.

Because VIA is designed primarily for local area networks, a front-end server is added to the VISA system to implement remote access. The front-end server supports the iSCSI protocol. As an inexpensive network storage protocol, iSCSI is widely used in SAN and some other network storage systems. In Internet, any storage server and device supporting the iSCSI protocol can process storage operations mutually.

vSCSI (VI-attached SCSI) protocol is designed and implemented to support SCSI commands in the VI network. The VISA storage system is based on vSCSI.

To compare with iSCSI and IP SAN, a prototype system is built and some results are obtained by testing with representative benchmarks.

Our testing environment consists of a 1Gbps Fast Ethernet connecting a number of nodes. Every node has a 3 SysKonnect SK-9821 100M/1000M Fast Ethernet adaptor to support M-VIA. The operating system running on each node is RedHat Linux 7.3 Kernel 2.4.18.

Fig.7 shows the performance comparison between vSCSI and iSCSI in transferring different block-sized packets. For sequential write/read, the throughput of vSCSI is markedly higher than iSCSI. In particular, the maximum achieved throughput with iSCSI is about half of the vSCSI. For random read/write, when block size is small, the vSCSI performance is worse than that of iSCSI by about 10%. But for large block size, vSCSI outperforms iSCSI, especially in the write mode.

Table 1 shows the performance comparison between VISA and IP SAN that are connected with the same Ethernet adaptor. The performance is nearly the same for random read/write. But for sequential write/read, the VISA performance is significantly better than that of IP SAN.

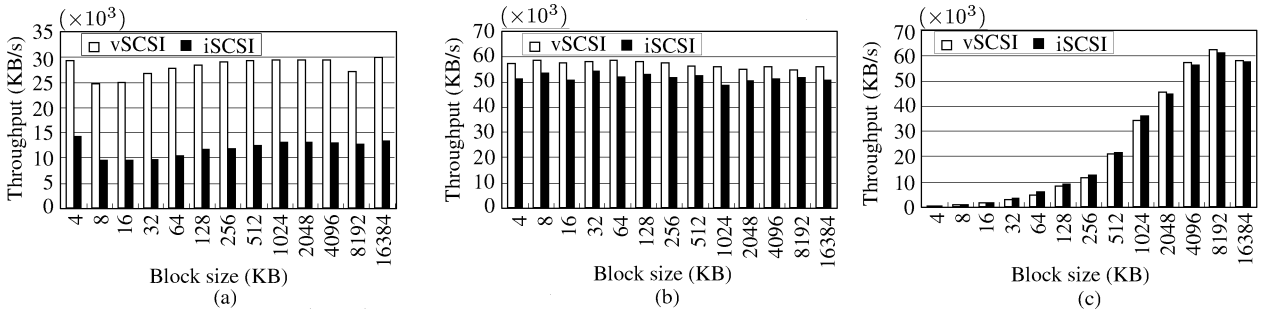

Fig.7. Comparison of throughput between vSCSI and iSCSI. (a) Sequential write. (b) Sequential read. (c) Random read/write.

Table 1. Performance of VISA and IP SAN with Different Number of Storage Nodes

	Sequential Write		Sequential Read	
Number of Storage Nodes	VISA (KB/s)	IP-SAN (KB/s)	VISA (KB/s)	IP-SAN (KB/s)
2	41556.18	17188.55	66216.76	43640.48
3	44759.53	20748.39	64327.28	44623.66
4	51747.68	28963.93	62531.04	46358.68

4.2 CEFT-PVFS

Disks (IDE or SCSI) in nodes of a cluster can potentially form a distributed disk array to meet the demand for large scalable storages. Parallel file systems, such as GPFS[28] and PVFS[31], have been developed to provide high I/O performance.

Without any additional cost, all the disks on the nodes of a cluster can be connected together through CEFT-PVFS[32], an RAID-10 style parallel file system, to provide a multi-GB/s parallel I/O performance. I/O response time is one of the most important measures of quality of service for a client. When multiple clients submit data-intensive jobs at the same time, the response time experienced by the user is an indicator of the power of the cluster.

4.2.1 Design and Implementation of CEFT-PVFS

CEFT-PVFS is an extension of PVFS, which aims to providing a multi-GB/s parallel I/O performance with significant fault-tolerant capability. There are compute nodes and I/O server nodes in the cluster, and a node can be either or both depending on the overall workload distribution. All I/O server nodes are divided into two groups, the primary one and the mirroring one. File data is striped across the primary group and duplicate on the mirroring group. When writing, data are stored in the primary group in the RAID 0 style and backed up in the mirroring group simultaneously. Data is retrieved from the nodes that have less workload between the mirrored pair to optimize the write performance.

The metadata manager handles operations such as file access permission, file size, striping size, and striped data location on disks. It only informs the clients the locations of the CEFT-PVFS file and does not participate in the read/write operations to avoid becoming a bottleneck of the system. Clients handle all file I/Os without the manager's intervention. Four different protocols for data duplication with different performance and reliability trade-offs are implemented to optimize performance for different application environments. A scheduling algorithm for dynamic load-balancing based on CPU, memory and disk workload is developed to improve the performance[32]. Compute nodes can directly access I/O servers through the network. Clients can be distributed in all nodes of the cluster.

4.2.2 Performance Analysis

The queuing model for the CEFT-PVFS service under data-intensive load is shown in Fig.8. Assume that I/O requests follow a Poisson process; with a mean arrival rate of λ. Each request is handled by the meta-data server to get meta-data, and then performs actual data read or write operations to I/O nodes. Part of the main memory space in a server node is used for I/O cache buffer to hide the disk I/O latency and to take advantage of data reference locality. Assume the number of I/O server nodes to be N in each group. The arrival rate to the server node i is $P_i\lambda$, where P_i is the probability that the request is directed to node i. When the I/O request is a small read or write, where the data size is equal to or less than the size of a striped block, and the workload on a server node among a group is balanced, P_i is equal to $1/N$. When the I/O request is a large read or write, where data is striped on all of the nodes in a group, P_i is equal to 1. So the typical range of P_i is $[1/N, 1]$. Let P_r and $P_w = 1 - P_r$ denote the read and write probability of a request, respectively. Assume that the I/O cache buffer hit rate for read requests to be h_r and the probability of write buffer being full to be f_w. Thus, the effective arrival rate to each disk is: $\lambda_i = [P_r(1 - h_r) + P_w f_w] \cdot P_i\lambda$.

Assume that the metadata service time, the network service time and the I/O buffer service time are exponentially distributed with the average times T_{MS}, T_{net} and T_c, respectively. Therefore, request residence time in the network and in the I/O buffer can be modeled using the M/M/1 queuing model[14]. According to the M/M/1 model[14], the average residence time in meta-data server can be calculated as:

$$W_{\mathrm{MS}} = \frac{T_{\mathrm{MS}}}{1 - \lambda \cdot T_{\mathrm{MS}}}. \quad (1)$$

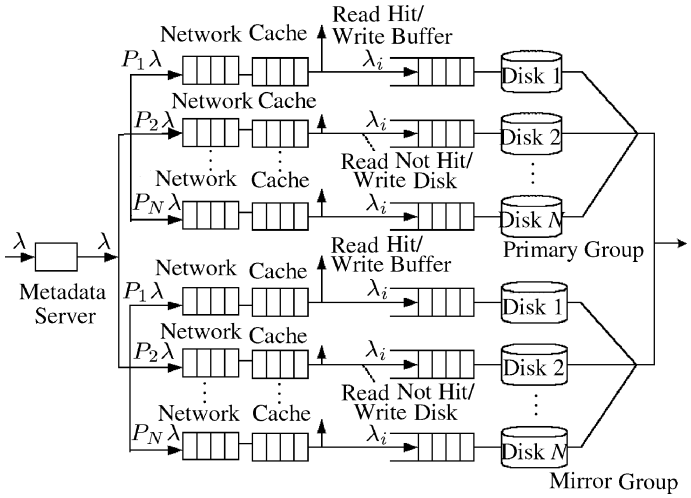

Fig.8. A queuing model of I/O service for CEFT-PVFS.

The average residence time in network can be calculated as:

$$W_{\text{net}} = \frac{T_{\text{net}}}{1 - P_i\lambda \cdot T_{\text{net}}}, \tag{2}$$

and the average residence time in the I/O buffer can be calculated as:

$$W_{\text{cache}} = \frac{T_c}{1 - P_i\lambda \cdot T_c}. \tag{3}$$

According to M/G/1 model[14], the average residence time in a disk drive can be calculated as:

$$W_{\text{disk}} = \frac{\lambda_i E(Y^2)}{2[1 - \lambda_i E(Y)]} + E(Y). \tag{4}$$

The average I/O response time is composed of three components, namely, the average transfer time for the network, the average residence time in the I/O buffer and the average residence time in the disk drive. So the average I/O response time can be expressed as:

$$\begin{aligned} Z &= W_{\text{MS}} + W_{\text{net}} + W_{\text{cache}} + P_{\text{disk}} \cdot W_{\text{disk}} \\ &= \frac{T_{\text{MS}}}{1 - \lambda \cdot T_{\text{MS}}} + \frac{T_{\text{net}}}{1 - P_i\lambda \cdot T_{\text{net}}} + \frac{T_c}{1 - P_i\lambda \cdot T_c} \\ &\quad + [P_r(1 - h_r) + P_w f_w] \times \left\{\frac{\lambda_i E(Y^2)}{2[1 - \lambda_i E(Y)]} + E(Y)\right\} \end{aligned} \tag{5}$$

where $T_{\text{net}} = L/R_{\text{net}}$, $T_c = L/R_{\text{memory}}$ and L is the size of data to be accessed on each I/O server node. Even though the data is striped in fixed blocks to server nodes in a RAID 0 style, the blocks can be incorporated into a large block with length equal to L. R_{net} and R_{memory} are the available network bandwidth and the available memory access rate respectively.

The available network bandwidth is about 120MB/s in the PrairieFire system and the memory access rate is about 500MB/s. The data striping block size is 64KB in CEFT-PVFS. Typical values for relevant disk parameters are: 50MB/s for data transfer rate, 1.2ms for track-to-track seek time, 8.5ms for average seek time, 4.17ms for average latency, and 19036 for the number of cylinders. T_{MS} is the time for a request handled by metadata server to get metadata and the average time is measured about 5μs in the PrairieFire system.

The results reveal that the response time is a function of several operational parameters. The results show that I/O response time decreases with the increase in I/O buffer hit rate for read requests, write buffer size for write requests and number of server nodes in the parallel file system, while higher I/O requests arrival rate increases I/O response time. On the other hand, the collective power of a large cluster supported by CEFT-PVFS is shown to be able to sustain a steady and stable I/O response time for a relatively large range of the request arrival rate.

5 GDSS: Global Distributed Storage System

Advances in science and technology are made possible largely through the collaborative efforts of many researchers in a particular domain. We see collaborations of hundreds of scientists in areas such as gravitational-wave physics[33], high-energy physics[34], astronomy[35] and many others coming together and sharing a variety of resources collaboratively in pursuit of common goals. These resources are geographically distributed and can encompass people, scientific instruments, computer and network resources, applications and data. It is common to see datasets on the order of terabytes today, and on the petabyte-scale soon. Grid technologies[36–38] enable efficient resource sharing in collaborative distributed environments.

One challenge facing data-intensive and high-performance computing applications is to provide efficient management mechanism and transfer model for terabytes or petabytes of information in wide-area, distributed environments[37,39]. It also needs efficient security mechanism to insure data confidentiality and integration.

GDSS is a novel architecture for a wide-area distributed storage system built on SAN, NAS, or any other storage systems. Our purpose is to construct a distributed storage system with high scalability, security, and efficiency to offer a high quality storage service to millions of users in the grid environment[36].

We propose a user and group-based multi-namespace architecture, and develop a new approach to solving the bottleneck problem of metadata server. A new component, called *Storage Service Provider* (SSP), is introduced to provide storage service for users, which play the role of user agent to the storage system. In order to distinguish QoS for users, files can be replicated, clipped, and stored in different storage devices, and the access is transparent for users.

5.1 A Global View of GDSS

GDSS is a middleware to unify heterogeneous storage resources to provide a huge available storage capacity

for millions of users. As described in Fig.9. It contains the following modules: *Storage Service Provider* (SSP), *Global Name Server* (GNS), *Resource Manager* (RM), *Storage Agent* (SA), and *Certification Authority* (CA).

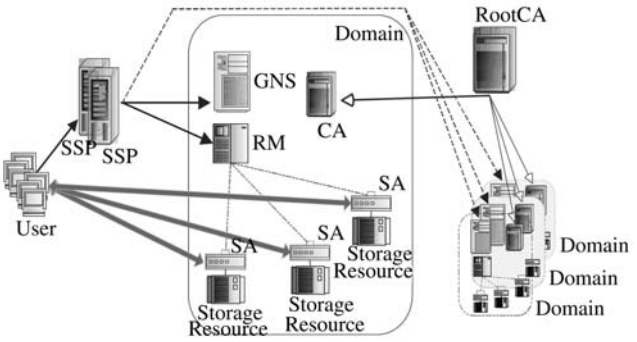

Fig.9. Structure of global distributed storage system.

SSP is the access point of the system, and the whole system is divided into several domains. In each domain there are GNS and RM to store metadata and replica information respectively, and to manage storage resources in the domain. There exists several SSPs in the system and each of them is equal in their status and capacity. SSP is the entry of the system through which all the system components are accessed. GNS is in charge of metadata management, including metadata operation interface, metadata fault-tolerant subsystem and metadata search engine. GNS contains several *Name Servers* (NS) that record parts of the metadata information and are organized as a tree-like structure to provide metadata information to other system components. RM maintains a storage resource list: it is in charge of resource application and scheduling. RM presents a transparent and dynamic replica scheme that can greatly reduce the access latency and bandwidth consumption and improve load balancing and reliability. The dynamic replica scheme automatically performs replica creation, deletion and management operations according to the condition of information access. CA is the certificate management subsystem: it enforces information access control, ensures system security, and records user's information. SA provides a standard access interface above heterogeneous storage systems and implements a data transfer protocol that can substantially improve the data transfer speed.

GDSS implements storage virtualization from three aspects: 1) SSP virtualizes distributed storage resources as a logical storage pool and provides a single storage image for users; 2) SA provides a uniform access interface to heterogeneous storage resources and ensures that GDSS adapts to the variety of storage resources; and 3) NS reflects the relationship between the logic storage pool and the physical storage devices.

5.2 Characteristics of GDSS

5.2.1 *Metadata Organization and Management*

GDSS transforms heterogeneous distributed storage resources into a big logic storage pool for users who do not care where the data is stored and how to fetch them. In order to attain the above goals, GDSS should manage all the data, user and group information, which is called metadata. The metadata server in GDSS is implemented by GNS that stores and manages metadata and makes metadata server extendable and fault tolerant.

GDSS uses LDAP to store all the metadata and organizes them as a tree-like structure. Metadata organization is separated into two branches: one records user and group information; the other records user or group file structure. Fig.10 illustrates an example of such organization with two users and one group. GDSS reflects the tree to every user and group to shape the user's single file image to which file access control is attached.

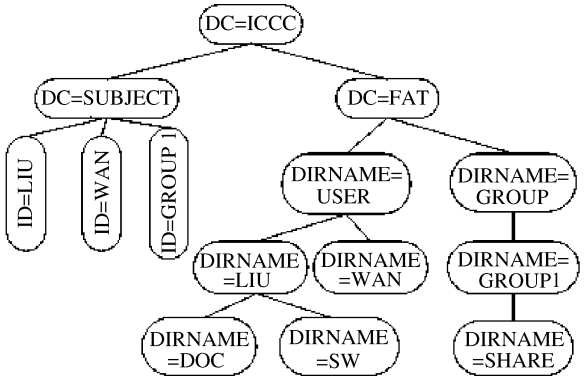

Fig.10. Metadata organization in GDSS.

With the increase in access frequency to GNS and in the number of users, metadata server should be dynamically extended to balance the access load. All the metadata is stored in NS managed by GNS in GDSS. NS can be extended to distribute metadata. Utilizing the referral defined in the LDAP protocol GDSS divides the whole metadata tree into several NSs to decrease the access frequency to every NS, as shown in Fig.11. A matching table is stored in GNS to indicate where to find the corresponding metadata information. Table 2 is an example matching table. When inquiring, GNS first checks its matching table and finds which NS stores it, and then fetches the metadata information through the corresponding NS.

Table 2. Example for Matching Table

Root Path	Name Server
/root	NS-A
/root/$B1$	NS-B
/root/$A1$/$A2$/$C11$	NS-C
/root/$B1$/$B2$/$B3$/$B4$/$D1$	NS-D

GDSS uses metadata replica to ensure the reliability of metadata. If the fault is a disk error or server outage, the system reallocates the space to store the corresponding metadata from its replica. The higher the security level of metadata, the more replicas will be made. Replica is stored in another NS. Through the matching table, system can find the information.

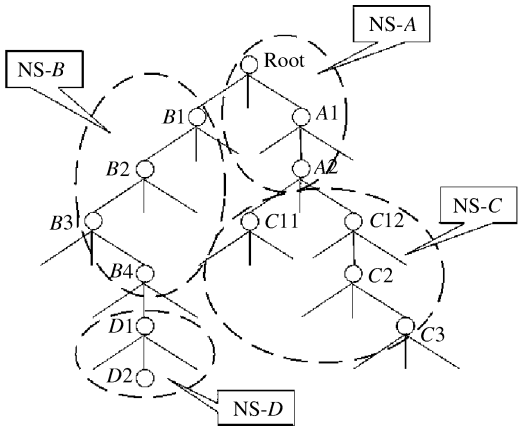

Fig.11. Example for metadata tree division.

5.2.2 Global Information Sharing

One of the main goals of GDSS is to enable the sharing of a wide range of information by users. There are two methods to provide information sharing: the exchange of private information between two users and the sharing of information through a shared storage space among for a large number of users. The main problem caused by information sharing is access control: different user has different right to a file and when a user access a file, system must limit the user's operation based on his/her access right.

Individual and group users are the two types of users defined in GDSS. An individual user can join one or more group users. A group user can join another user group. The relationship between individual user and group user is shown in Fig.12.

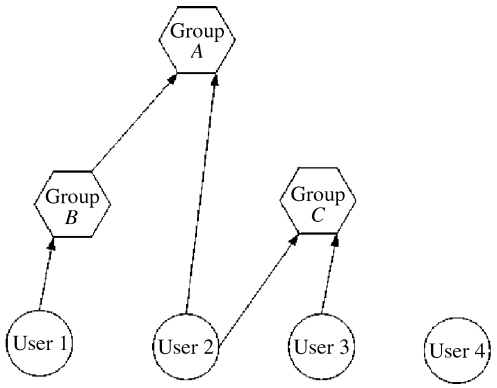

Fig.12. Relationship between group and individual user.

For an individual user, access right does not need to be attached to directories in its own name space and only this user can access directories in his/her own name space. An individual user can share resource through output sharing that is associated with access right. Other users can access the resource if the access right is granted. When an individual user joins a user group, he/she can access the corresponding resource according to the role is assigned by the group administrator. Every directory in group user name space has an access control list and lower lever directory inherits the access rights from the upper lever directory.

5.2.3 Data Transfer

Data transfer is the main function of GDSS that largely determines the system performance. GDSS modifies several aspects of traditional data transfer protocol to make it more efficient.

Sliced-data transfer is to slice data into several pieces and get the data from different storage devices. Since there are several data replicas in GDSS, the system can get the data from different storage devices and each device provides a piece of data.

Another modification is the socket channel. The speed of small-size file transfer is very low, and the transfer of a batch of small-size files is about 2–15 times slower than that of a single file with the same size when using FTP[40,41] as the transfer protocol. Socket channel is used in GDSS to optimize small-size file transfer. Socket channel encapsulates a batch of small-size files into one socket command, in which one socket channel is shared by all the files. The data structure of socket channel is shown in Fig.13.

Establish Connection	File Info	File Data	File Info	File Data	...	Close Connection

Fig.13. Style of data channel.

Fig.14 shows the operation flow of the datatransfer subsystem in GDSS. First, the user issues an operation request, then according to the operation type and metadata, the data transfer subsystem creates transfer tasks and put them into a task queue. The transfer controller fetches transfer tasks from the task queue and submits it to the lower data transfer layer to complete data transfer. During the transfer process, the status of transfer will be fedback to the transfer task. We use this information to monitor the transfer process. When transfer completes, the transfer controller confirms the usage of resource and modifies the metadata.

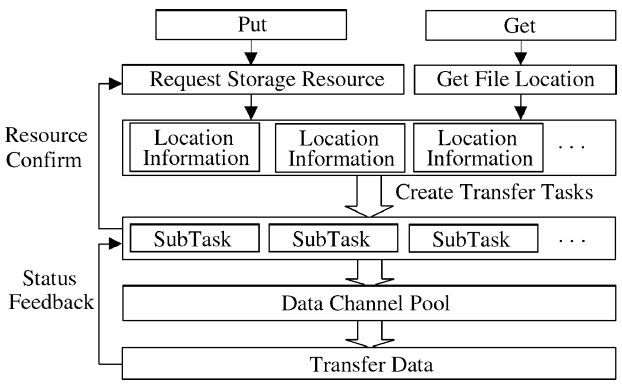

Fig.14. Operation flow of transfer subsystem in GDSS.

5.2.4 Security in GDSS

In GDSS the basic security infrastructure provides the following functions: 1) *Secure communication.* Mutual authentication happens before data transfer. During data transfer the data can be encrypted and integrity can be guaranteed; 2) *Security across organizational boundaries.* There may be many security domains and all the domains can coordinate to provide a distributed and manageable security system; 3) *Single sign-on.* In order to support the mobile users, the system provides a single sign-on so that the user can access the system anywhere via any SSP; 4) *Use-defined security class.* Users can define the security class to reduce some unnecessary overhead.

5.3 Operations of GDSS

GDSS provides an ftp client and a special client to access data. Using the special client the user attains high performance of data transfer because it uses the improvements in SA. Using the ftp protocol a user cannot attain those characteristics.

Fig.15 shows the process of the read file process through the APIs of the system: 1) a client sends a connect command to an SSP with user name, group name and password; 2) the SSP passes the connect request to a CA, the CA verifies the information. If the current CA could not verify this user, it sends the verification request to another CA. For a legitimate user, a user information table will be returned; 3) the user sends the read command with the file names to the SSP. The SSP finds out which metadata server keeps the metadata of the files from the Metadata scheduler. The SSP sends the file names to the corresponding metadata server and gets the metadata to the client; 4) with the metadata the client can easily get data from Agents.

In some cases, a user has no special clients that can use the API defined by the system. When a user accesses the system through standard FTP clients or HTTP clients, the operation procedures are different from the above. SSP does substantial work for the client. It must be noted that when the file is divided the file must be merged through SSP, the data transfer will be through SSP. Otherwise the third part of transfer will be adopted as shown in Fig.16.

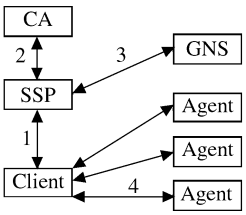

Fig.15. Read operations through API of GDSS.

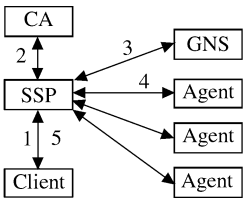

Fig.16. Read operations through general client.

5.4 Performance Evaluation

Fig.17 shows system extendibility. The evaluation environment consisted of 100M Ethernet and disks with IDE interface. Several users transfer different 100M-size files simultaneously. We conclude that system collective bandwidth increases linearly with the increase of storage resources, which indicates that GDSS has good extendibility.

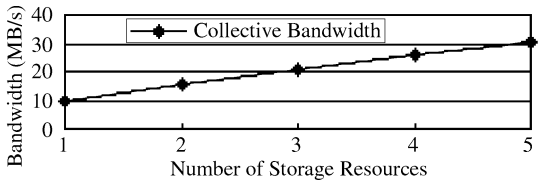

Fig.17. Relationship between system bandwidth and storage resource number.

Fig.18 shows a comparison between the traditional FTP server and the socket channel when transferring a batch of small files. In the experiment, 100 files of the same size are transferred, and the average transfer time is obtained.

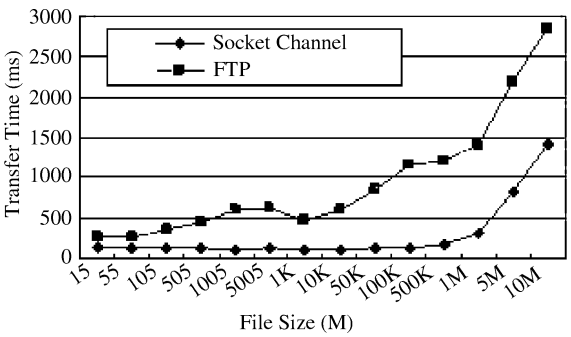

Fig.18. Comparisons between socket channel and traditional FTP when transferring a batch of small files.

Fig.18 shows that the time of transferring a batch of small files (whose size is less than 500K) changed slightly using standard FTP. Frequent connections between servers and clients can explain this phenomenon. When the size of files increase from 500K, the transfer rate rises gradually. The transfer rate of the socket channel is 1–8 times as fast as that of the standard FTP. Fig.18 also shows that the average transfer time for files whose size is less than 500B is larger than that of transferring files whose size is between 1K–100K. Reasons for this phenomenon are redundant file information and frequent file operations (such as open and close). It can be predicted that performance improvement will be more remarkable as network latency increases.

Fig.19 shows a comparison between FTP and GDSS when transferring large files. Because of the network bandwidth and the processing limitation of clients, the transferring bandwidths of servers are limited to 1500KB/s. To simplify the implementation of the prototype system, we slice files according to its size. That is, no slicing is done if the file size is between 50–100MB, 2

slices for the file size between 100–200MB.

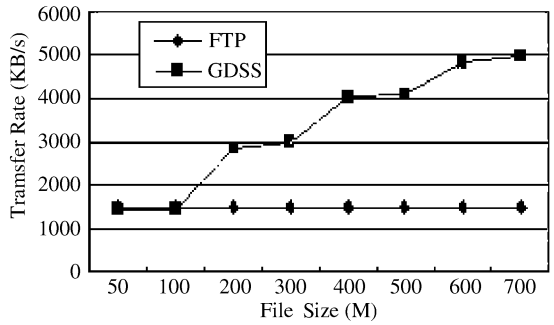

Fig.19. Comparisons between GDSS and traditional FTP when transferring large files.

Fig.19 shows that as file size increases, the transfer rate increases and the transfer rate using GDSS is much higher than that of the standard FTP.

6 Tape Virtualization Technology

Virtualization is charting a new direction for the technological development of storage systems. With the help of the virtualization technology, a *Tape Library File System* (TLFS) is proposed[42]. Also, virtual disk, which looks like a massive disk, maintains a consistent view of massive tape and RAID storage, so that the user can effectively manage them.

6.1 Tape Library File System

The primary benefits for the end users are to reduce primary disk space and improve performance (as expressed in faster response time). However, for the industries, the first type of disk option, simply buying an inexpensive disk array and backing up to a file system, has some limitations. For instance, the backup software may require a license to be used in a file-system-type device, such as a RAID, backing up to a file system is more complicated and costly in management than backing up to tape (or tape library). General file systems are more prone to being infested with viruses, and thus have an inherent problem of fragmentation.

In an attempt to address the above problems, TLFS effectively integrates the *Virtual Tape Library* (VTL)[43] technology, the RAID-DP technology[44], and the iSCSI technology to provide transparent tape file access for users while retaining other functions of tape libraries. Our study of TLFS shows the main advantages of TLFS over the conventional tape libraries and conventional (disk-based) file systems as follows: 1) high backup and restoration performance; 2) low cost compared with simple disk-based systems; 3) some finer functions integrated both the disk and the tape library.

TLFS allows users to create files and directories as well as delete, open, close, read, write and/or extend the files on the device(s). TLFS maintains security on the files and provides the management for fragmentation. Moreover, TLFS can support large-scale file systems. Users have two ways to access TLFS: using general backup application, and through the APIs provided by TLFS.

In the target, our function modules are implemented in SCST[45], which is a generic SCSI target middle level for Linux. It is designed to provide a unified, consistent interface between SCSI target drivers and Linux kernel and simplify target driver development as much as possible. Although the data distribution policy is different compared with [46, 47], in substance, their implementation technologies are analogical. And they all have to record all the logical objects[48] on the RAID.

The SCSI command analysis module receives SCSI sequential commands from the backup application, and determines whether the commands should be executed on the RAID or on the tape library. Then it delivers them to the proper module (or media). The SCSI command transform module is responsible for transforming SCSI sequential commands into SCSI block commands. The LBA (*logical block address*) mapping module maintains the block mapping information, which associates the logical unit of an object with its logical block address in the RAID. The data transfer module performs data transfer between RAID and tape library according to some information lifecycle management policy. Tables 3 and 4 show the results.

Table 3. Average Write/Read Time of the VTL Tape in Different Network Environment

Number of Files	Total Size of File (MB)	VTL Tape in TLFS			
		1000M Ethernet		100M Ethernet	
		Write (s)	Read (s)	Write (s)	Read (s)
1	50.0	2	3	3	4
1	200.0	10	17	12	21
1	1000.0	53	84	55	106
381	85.0	6	5	6	6
3386	455.1	39	29	40	30
22216	1000.0	112	79	112	91
10	2000.0	238	247	259	251

The average write time of the VTL tape in 1000M Ethernet is 99.14% of that in 100M Ethernet and the average read time in 1000M Ethernet is 95.68% of that in 100M Ethernet, which indicated that different network environment has few influence for TLFS.

Table 4. Performance Comparison of the Local Node of VTL and the Physical Tape

Number of Files	Total Size of File (MB)	VTL in Local Node		Physical Tape	
		Write (s)	Read (s)	Write (s)	Read (s)
1	50.0	2	2	8	17
1	200.0	9	16	32	59
1	1000.0	47	80	163	305
381	85.0	5	4	16	24
3386	455.1	27	20	87	119
22216	1000.0	84	52	132	223
10	2000.0	92	112	268	330

It indicates network has large influence for TLFS. In the same time, the average write time of the VTL tape in local host is 54.62% of that in 100M Ethernet and the average read time in local host is 61.64% of that in 100M Ethernet

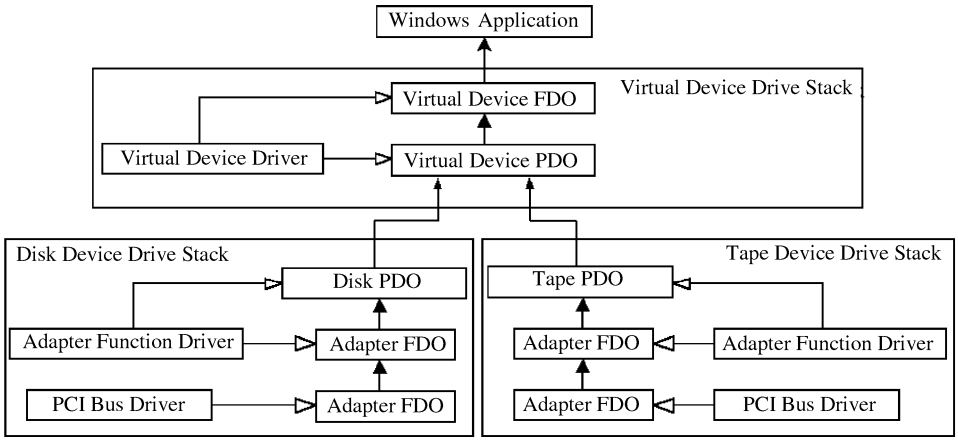

Fig.20. Virtual device drive stack of Microsoft Windows.

6.2 Virtual Disk

Any problem in computer science can be solved with another layer of indirection[49]. Virtual disk, which is implemented both under Redhat Linux (kernel 2.4.20-8) and under Microsoft Windows 2000, is a hybrid storage system based on RAID and tape library. For users, it looks like a general disk, except for its very large capacity (equal to the sum of all tapes' capacity).

In Microsoft Windows 2000 development environment, a virtual device driver has both virtual device PDO (*physical device object*) and FDO (*functional device object*), shown in Fig.20. This virtual device looks like a disk by a Windows application. As shown in Fig.21, this design is built above the disk device drive stack and the tape device drive stack, so users do not have to care about different vendor's storage device driver. Additionally, for virtual disk, the real access address space is the sum of RAID's and all tapes' address space.

7 Domain-Based Storage Management Framework for the Heterogeneous WAN-Based Environment

As the Internet continues to grow and the storage technologies evolve, users want to get storage services at anytime and anywhere by connecting to the network. A domain-based storage management framework is designed for the Wide Area Network (WAN) toward catering for the large-scale and heterogeneous storage environment. It is based on the Storage Management Initiative Specifications (SMI-S).

7.1 Storage Management Framework Architecture

The framework can be generally divided into 3 major components, which are storage system agent, domain agent and management application. The architecture is illustrated in Fig.21.

Users can access one of multiple administrative domains through the management application, and the protocol between the management application and the domain agents is XML/HTTP to which the CIM operates over HTTP mechanism of SMI-S that is applied. In a domain, StorageSys agent represents the storage system and domain agent represents the sub-domain, they can communicates with the domain agent for registration and transmitting information through SIP and XML/HTTP. A StorageSys agent can access storage system resources with a specific interface such as protocols, files, etc. We shall show each of these components in more details.

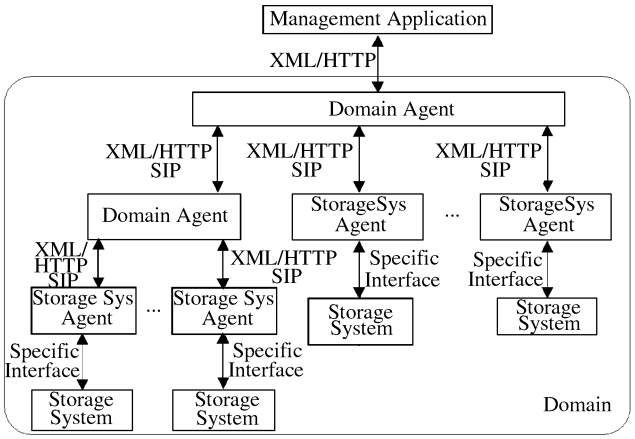

Fig.21. Overall management framework architecture.

7.1.1 *StorageSys Agent*

The StorageSys agent has two main responsibilities for the storage system: 1) allowing the storage system to dynamically register to be a member of a certain domain, and 2) collecting, dealing with and transmitting management information of the storage system. The agent consists of one common information model object manager (CIMOM), one SIP provider and a set of storage resource providers (SRPs). Fig.22 shows the architecture of a StorageSys agent.

CIMOM is a central component that routes and saves information about storage resources and events objects

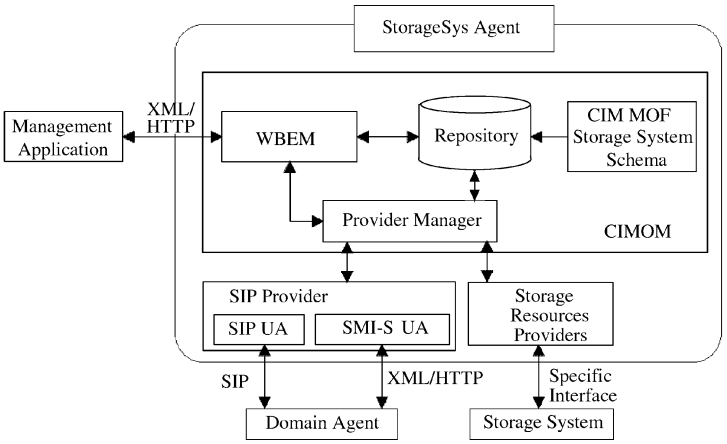

Fig.22. Architecture of StorageSys agent.

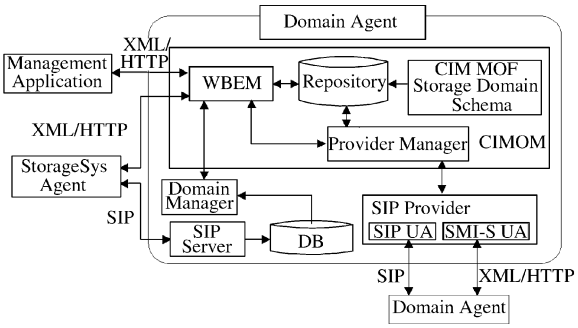

Fig.23. Architecture of domain agent.

between components. The storage resources and events objects are described by SMI-S-compliant CIM classes and CIM instances and stored in CIM repository. The CIMOM responds to the operations from management application defined in the WBEM specification, such as create, modify, and delete. It also interacts with providers, which actually obtain the information from the resources, through the Provider Manager.

Each SRP corresponds to one storage resource within the storage system, and via a specific interface, such as SNMP, RPC, etc., collects, gathers and updates the storage resource information. SRP is in fact a so-called instrumentation agent to obtain the information from the resources and forward it to the CIMOM.

SIP Provider is used to implement StorageSys agent registration by means of one function module SIP UA (user agent), then send the storage system common information objects into the domain with the other function module SMI-S UA if the registration has been successful.

7.1.2 *Domain Agent*

The domain agent plays the role of a manager in a domain, which consists of one CIMOM, one domain manager, one SIP Server with DB, and one SIP Provider. Fig.23 illustrates the architecture of the domain agent. The domain agent can allow its own domain freely become one sub-domain of another domain, by means of SIP Provider, which improves the scalability of the framework.

SIP Server is used to achieve the goal of automatic discovery of storage system. SIP server receives registration messages from other agents that would like to join the domain, parse these messages and store the entities' location information into DB.

Domain manager is responsible for domain information statistics. Due to the fact that the managed objects and information frequently change due to dynamic registration actions, it is necessary for the domain agent to collect statistics on information such as total domain storage capacity, spare storage capacity, etc. It executes the management process on the basis of data from DB and repository in CIMOM. With respect to what way to run the task, we periodically run it at present. The policy-based and triggered way is the future pattern we will adopt.

7.2 SMI-S Based Storage Management Scheme

7.2.1 *Storage System Management Scheme*

We can sufficiently describe the management resources information of the storage system via SMI-S profiles[50], which are based on DMTF's CIM core and common models[51], such as fabric profile, array profile, etc. The management objects can be described in XML as well as the Management Object Format (MOF)[52]. We also use the corresponding SRPs (See "StorageSys Agent" in Subsection 2.2) to obtain the actual resources' information. Fig.24 shows the storage system management scheme.

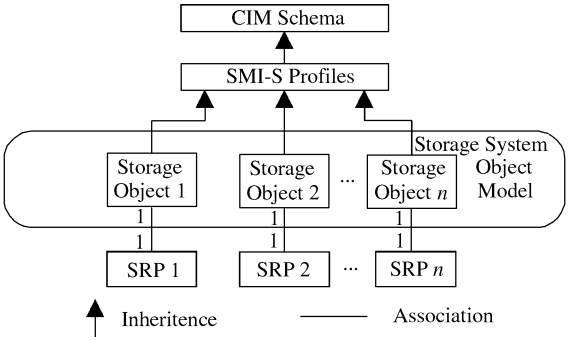

Fig.24. Storage system management scheme.

The StorageSys agent will collect and deal with the dynamic resource information through SRPs once it receives the WBEM-compliant management requests or registers successfully.

7.2.2 *Storage Domain Management Scheme*

For the domain management scheme, first, we also create some managed objects that are derived from the CIM objects defined in SMI-S profiles and represent the

storage domain common information, such as total domain storage capacity, spare storage capacity, etc. These objects are called Common Information Objects (CIOs).

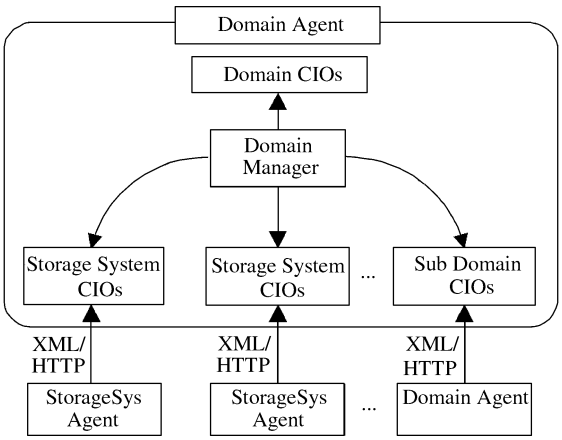

Fig.25. Storage domain management scheme.

The system agent or other domain agent will integrate its CIOs including CIM classes and instances into the CIM repository of the domain agent, through SMI-S UA of the SIP provider, which uses schema or instance manipulation operations such as CreateClass, ModifyClass, DeleteClass, CreateInstance, ModifyInstance, and DeleteInstance[53], after it registers successfully. So, the domain manager is designed to periodically collect and deal with these existing objects' information, and update domain's CIOs' attribute value. Fig.25 illustrates the process.

7.3 Implementation

A prototype is implemented that consists of two StorageSys agents, one domain agent and one Web-based management application. The two StorageSys agents are respectively responsible for a RAID located in Wuhan and a SAN housed in Beijing, China. The domain agent is also in Wuhan, which controls the two StorageSys agents, meanwhile, we deploy the management application in the same place with domain agent. It can manage the storage system in WAN and it is shown to be scalable.

8 Conclusion

Applications such as e-Science and e-Business involve geographically distributed and heterogeneous resources such as computational resources, scientific instruments, and databases. The data in these applications is usually massive and distributed across numerous institutions for various reasons including the inherent distribution of data sources; large-scale storage and computational requirements; to ensure high-availability and fault-tolerance of data; and caching to provide faster access. To satisfy such mounting demand, storage is required to be more scalable, reliable, secure, manageable and intelligent. Building, managing, and operating such distributed data and storage systems in autonomic manner are not only challenging, but also presents a large business opportunity for storage industry.

We present some of these achievements from several storage systems under development: a storage pool with network-attached RAID, an object storage system, a high performance distributed storage system for local area network storage, and a global distributed storage system for wide area network storage. All of these storage systems are online massive storage systems. A typical offline massive storage system is used to backup data or store documents, for which the tape virtualization technology is described in the paper also. All these types of massive storage system can meet different demands for huge data storage.

References

[1] Gibson G A, Patterson D A. Designing disk arrays for high data reliability. *Journal of Parallel and Distributed Computing*, Jan, 1993, 17(1): 4–27.

[2] Gibson G. Scaling file service up and out. In *Keynote Address at the 3rd USENIX Symp. File and Storage Technologies* (*FAST'04*), San Francisco, California, March 31, 2004.

[3] Braam P J. The Lustre storage architecture. Technical Report, Cluster File Systems, Inc., January 2004, http://www.lustre.org/docs/lustre.pdf.

[4] Object Storage Architecture. White paper, January 2004, http://www.panasas.com/activescaleos.html.

[5] Intel Corporation. Object-based storage: The next wave of storage technology and devices. Jan. 2004, http://www.intel.com/labs/storage/osd/.

[6] Tierney B, Johnston W, Lee J *et al.* A data intensive distributed computing architecture for grid applications. *Future Generation Computer Systems*, April 2000, 16(5): 473–481.

[7] Baru C, Moore R, Rajasekar A *et al.* The SDSC storage resource broker. In *Proc. CASCON'98 Conf.*, Toronto, Canada, 1998.

[8] Rhea S, Eaton P, Geels D *et al.* Pond: The OceanStore prototype. In *Proc. the 2nd USENIX Conf. File and Storage Technologies* (*FAST'03*), March 2003.

[9] Allcock W, Bester J, Bresnahan J *et al.* GridFTP protocol specification. *GGF GridFTP Working Group Document*, September 2002.

[10] Riedel E, Faloutsos C, Gibson G A, Nagle D. Active disks for large-scale data processing. *IEEE Computer*, June 2001, 34(6): 68–74.

[11] Mesnier M, Thereska E, Ellard D *et al.* File classification in self-*storage systems. In *Proc. the First Int. Conf. Autonomic Computing* (*ICAC-04*), New York, NY. May 2004, pp.44–51.

[12] Kephart J O, Chess D M. The vision of autonomic computing. *IEEE Computer*, 2003, pp.41–50.

[13] Storage Networking Industry Association. CIM-SAN-1 Vendor Profile, Feb.2006, http://www.snia.org/tech_activities/SMI/CIMSAN/emc.doc

[14] Sivathanu M, Bairavasundaram L, Arpaci-Dusseau A C *et al.* Database-aware semantically-smart storage. In *Proc. Fourth USENIX Symp. File and Storage Technologies* (*FAST'05*), San Francisco, California, December 2005, pp.239–252.

[15] Morris R. Storage: From atoms to people. In *Keynote Address at Conference on File and Storage Technologies* (*FAST'02*), Monterey, California, USA, January 28–30, 2002.

[16] Zhou K, Feng D, Wang F, Zhang J-L. Research on network RAID pipeline technology. *Chinese Journal of Computers*, Mar. 2005, 28(3): 319–325. (in Chinese)

[17] Farley M. Building Storage Network. 2nd Edition, Osborne/McGraw-Hill, May 22, 2001.

[18] Zeng L, Feng D, Zhou K, Wang F. Research on mechanism of network storage I/O pipelining. *Mini-Micro Systems*, 2006, 27(1): 42–45. (in Chinese)

[19] Wang F, Zhang J-L, Feng D *et al.* Adaptive control in Heter-RAID system. In *Proc. Int. Conf. Machine Learning and Cybernetics*, 4–5 Nov. 2002, pp.842–845.
[20] Lampson B W. Hints for computer system design. *ACM Operating Systems Review*, October 1983, 15(5): 33–48.
[21] KleinOsowski K, Ruwart T, Lilja D J. Communicating quality of service requirements to an object-based storage device. In *Proc. 22nd IEEE/13th NASA Goddard Conf. Mass Storage Systems and Technologies* (*MSST 2005*), Monterey, CA, USA, April 2005, pp.224–231.
[22] Feng D, Zeng L. Adaptive policy triggering for load balancing. In *Proc. the 6th Int. Conf. Algorithms and Architectures*, Melbourne, Australia, October 2005, pp.240–245.
[23] Feng D, Qin L, Zeng L *et al.* A scalable object-based intelligent storage device. In *Proc. the 3rd Int. Conf. Machine Learning and Cybernetics*, China, Aug. 2004, pp.387–391.
[24] Wang F, Brandt S A, Miller E L, Long D D E. OBFS: A file system for object-based storage devices. In *Proc. 21st IEEE/12th NASA Goddard Conf. Mass Storage Systems and Technologies* (*MSST'04*), College Park, MD, April 2004.
[25] Morris J H, Satyanarayanan M, Conner M H *et al.* Andrew: A distributed personal computing environment. *Communications of the ACM*, Mar. 1986, 29(3): 184–201.
[26] Soltis S R, Ruwart T M, O'Keefe M T. The global file system. In *Proc. the 5th NASA Goddard Conf. Mass Storage Systems and Technologies*, College Park, MD, 1996.
[27] Satyanarayanan M, Kistler J J, Kumar P *et al.* Coda: A highly available file system for a distributed workstation environment. *IEEE Trans. Computers*, 1990, 39(4): 447–459.
[28] Schmuck F, Haskin R. GPFS: A shared-disk file system for large computing clusters. In *Proc. the 2002 Conf. File and Storage Technologies* (*FAST'02*), USENIX, Montery, CA, USA, Jan. 2002.
[29] Wang F, Zhang S, Feng D. A hybrid scheme for object allocation in a distributed object-storage system. In *Proc. Int. Conf. Computational Science 2006* (*ICCS'06*), UK, 2006.
[30] Chen J, Feng D. VISA: A virtual interface storage architecture for improved network performance. In *Proc. the 2nd Int. Conf. Embedded Software and Systems,* Xi'an, China, December, 2005, pp.587–592.
[31] Carns P H, Ligon W B III, Ross R B *et al.* PVFS: A parallel file system for Linux clusters. In *Proc. the 4th Annual Linux Showcase and Conf.*, Atlanta, GA, Oct. 2000, pp.317–327.
[32] Zhu Y, Jiang H *et al.* Design, implementation, and performance evaluation of a cost-effective fault-tolerant parallel virtual file system. In *Proc. Int. Workshop on Storage Network Architecture and Parallel I/Os*, New Orleans, LA, 2003.
[33] Barish B C, Weiss R. LIGO and the detection of gravitational waves. *Physics Today*, 1999, 52: 44.
[34] Wulz C-E. CMS— Concept and physics potential. In *Proc. II-SILAFAE*, San Juan, Puerto Rico, 1998.
[35] NVO, 2004. http://www.us-vo.org/.
[36] Foster I, Kesselman C, Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, Aug. 2001, 15(3): 200–222.
[37] Chervenak A, Foster I, Kesselman C *et al.* The data grid: Towards an architecture for the distributed management and analysis of large scientific data sets. *J. Network and Computer Applications*, 2001, 23(3): 187–200.
[38] Rajasekar A, Jagatheesan A. Data grid management systems. In *Proc. the 2003 ACM SIGMOD Int. Conf. Management of Data*, April 2003.
[39] Jin H, Cortés T, Buyya R (eds.). High Performance Mass Storage and Parallel I/O. IEEE Press, John Wiley & Sons, Inc., 2002.
[40] Postel J, Reynolds J. File transfer protocol (FTP). Oct. 1985. http://www.ietf.org/rfc/rfc0959.txt?number=959.
[41] Postel J, Reynolds J. Telnet protocol specification (FTP). May 1983. http://www.ietf.org/rfc/rfc0854.txt?number=0854.
[42] Feng D, Zeng L, Wang F, Xia P. TLFS: High performance tape library file system for data backup and archive. In *Proc. the 6th Int. Conf. High Performance Computing in Computational Sciences* (*VECPAR'2006*), Rio de Janeiro, Brazil, July 10–12, 2006.
[43] Anderson T E, Dahlin M D, Neefe J M *et al.* Serverless network file systems. *ACM Trans. Computer Systems*, Feb. 1996, 14(1): 41–79.
[44] Lueth C. NetApp data protection: Double parity RAID for enhanced data protection with RAID-DP. March 5, 2005, http://www.netapp.com/tech_library/3298.html
[45] Palekar A, Ganapathy N, Chadda A *et al.* Design and implementation of a Linux SCSI target for storage area networks. In *Proc. the 5th Annual Linux Showcase & Conference*, Oakland, CA, USA, 2001.
[46] Mu F, Shu J, Li B *et al.* A virtual tape system based on storage area networks. In *Proc. GCC'2004 Workshop on Storage Grid and Technologies*, LNCS 3252, 2004, pp.278–285.
[47] Myllymaki J, Livny M. Disk-tape joins: Synchronizing disk and tape access. *ACM SIGMETRICS Performance Evaluation Review*, 1995, pp.279–290.
[48] ANSI. SCSI Stream Commands-2 (SSC-2). Revision 09, July 9, 2003, http://www.t10.org.
[49] Wheeler D. Any problem in computer science can be solved with another layer of indirection. March 26, 2005, http://www.computer.org/computer/homepage/0505/GEI/
[50] Storage Networking Industry Association (SNIA). (2003). SNIA Storage Management Initiative Specification Version 1.0.1. Retrieved October 5, 2005, from http://www.snia.org/smi/tech_activities/smi_spec_pr/spec/SMIS_v101.pdf.
[51] Distributed Management Task Force (DMTF). (Jun 14 1999). Common Information Model (CIM) Specification Version 2.2. Retrieved October 7, 2005, http://www.dmtf.org/standards/cim/DSP0004.pdf.
[52] Distributed Management Task Force (DMTF). Management Object Format (MOF). http://www.dmtf.org/education/mof.
[53] Distributed Management Task Force (DMTF). Specification for CIM operations over HTTP Version 1.0, DMTF Specification, Aug. 1999.

Dan Feng received the Ph.D. degree from Huazhong University of Science and Technology (HUST), Wuhan, China, in 1997. She is currently a professor of School of Computer, HUST. Her research interests include computer architecture, storage system, parallel I/O, massive storage and performance evaluation.

Hai Jin received his Ph.D. degree in computer engineering from HUST in 1994. Dr. Jin now is a professor of computer science and engineering, the director of Cluster and Grid Computing Lab, and the Dean of School of Computer Science and Technology at HUST. He also is the chief scientist of the ChinaGrid project, one of the largest national grid computing in China. His research interests include computer architecture, cluster computing, grid computing, semantic web, peer-to-peer computing, network storage, network security, and pervasive computing.