Design and Implementation of an Out-of-Band Virtualization System for Large SANs

Guangyan Zhang, Jiwu Shu, Wei Xue, and Weimin Zheng

Abstract—Out-of-band virtualization intrinsically has the potential to provide high performance and good scalability. Unfortunately, existing out-of-band virtualization systems have some limitations, such as restrictions to specific platforms and/or hardware. In this paper, we present a new out-of-band virtualization system, MagicStore, which is not limited to any specific hardware and supports three widely used host platforms: Windows, Solaris, and Linux. First, MagicStore uses the SLAS2 approach to scale round-robin striped volumes efficiently. Second, it survives panics and power failures robustly through a combination of lazy synchronizations, ordered writes, and REDO logging. Third, it also incorporates typical legacy storage quickly by analyzing partition tables and reconstructing logical volumes. Our evaluation results from representative experiments demonstrated that MagicStore has the ability to provide high performance, to introduce low processor overhead, and to have good scalability.

Index Terms—Storage area network, out-of-band virtualization, scaling striped volumes, metadata integrity, legacy storage.

1 Introduction

1.1 Motivation

Storage virtualization [1] creates an abstraction of the storage resources, which allows a system to execute its I/O to blocks of generic storage, without needing to know exactly where or how the data is stored. It enables the competence of a logical volume to go beyond the limits of single physical devices. Therefore, virtualization can take full advantage of storage resources in a large storage area network (SAN) to provide higher quality of service (QoS), that is, larger storage capacity, higher I/O performance, and better data availability [2].

Storage virtualization in SAN environments is categorized into these groups based on the execution location: 1) Host-based virtualization: Most operating systems have their own volume managers. Host-based virtualization is not confined to specific hardware. However, it often brings larger overhead to hosts and does not support heterogeneous host platforms. 2) Storage-based virtualization: Many storage devices, such as RAID, provide virtualized logical unit numbers (LUNs). Storage-based virtualization has no negative impact on host performance. However, it makes virtualized devices limited in size and unable to cascade and, hence, impairs the scalability of storage devices. 3) Network-based virtualization: Virtualization occurs on the storage network between hosts and storage. Network-based virtualization both overcomes the scalability limitation of storage-based virtualization and offloads some of the work associated with virtualization from hosts.

According to whether storage virtualization sends control information in the form of metadata along the same path as the data transport or not, network-based virtualization is subcategorized into in-band virtualization and out-of-band virtualization. In the in-band architecture, the virtualization appliance itself is prone to becoming a bottleneck for storage transactions as the traffic load from multiple hosts increases. Conversely, out-of-band virtualization has the potential to provide high performance and good scalability since it places the appliance outside the primary path between hosts and storage.

There have already been several virtualization systems that use the out-of-band architecture. Examples of such systems include HP VersaStor [3], StoreAge SVM [4], and others [5], [6], [7]. Unfortunately, most of these systems suffer from some or all of the following limitations:

1. Platform dependence. Most out-of-band virtualization systems are customized for their own products by different vendors and, hence, are dependent on specific platforms and/or hardware. To the best of our knowledge, there is no out-of-band virtualization system that is implemented through pure software and that supports multiple heterogeneous platforms.
2. Scaling striped volumes. Scaling a round-robin striped volume can help both enhance the I/O performance and enlarge the storage capacity of a system. So far, however, most existing systems do not support online scaling. The few (for example, TH-VSS [7]) that include this function are inefficient and cannot be scaled for large-scale storage.
3. Incorporating legacy storage. For a data center where an out-of-band virtualization system will be deployed, it is very useful to incorporate a large legacy storage system rapidly. Existing systems can only copy the entire data on legacy volumes to virtualized volumes. As a result, the incorporating process is very time consuming.

• *The authors are with the Department of Computer Science and Technology, Room 8-201, East Main Building, Tsinghua University, Beijing 100084, PR China. E-mail: zhang-gy04@mails.tsinghua.edu.cn {shujw, xuewei, zwm-dcs}@tsinghua.edu.cn.*

Manuscript received 6 June 2006; revised 10 Jan. 2007; accepted 1 June 2007; published online 28 June 2007.
Recommended for acceptance by X. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0219-0606.
Digital Object Identifier no. 10.1109/TC.2007.70765.

1.2 Contribution

In this paper, we present and evaluate a new out-of-band virtualization system, MagicStore, which is not restricted within any specific hardware. Its virtualization appliance is implemented in Linux and virtualization agents are implemented for three platforms commonly found in data centers: Windows, Solaris, and Linux. This paper presents three major contributions of this new system:

1. Scaling round-robin striped volumes. Scaling a striped volume needs to redistribute almost all of the data chunks[1] on the volume online so as to preserve the round-robin order. MagicStore uses the SLAS2 approach, which takes advantage of the reordering window characteristic, to scale round-robin striped volumes efficiently.
2. Surviving panics and power failures. Some virtualization transactions require modification of virtualization metadata, which results in the challenge of keeping the metadata consistent. MagicStore can survive panics and power failures robustly through a combination of lazy synchronizations, ordered writes, and REDO logging.
3. Incorporating large legacy storage. MagicStore obtains the volume layout by analyzing partition tables on legacy disks and generates virtualization metadata to reconstruct a logical volume for each partition. Therefore, MagicStore can incorporate a large legacy storage quickly.

1.3 Road Map

The rest of this paper is organized as follows: Related work is the subject of Section 2. Section 3 presents the design of the MagicStore system. The implementation of MagicStore is described in Section 4. Section 5 evaluates the MagicStore system through detailed experiments. Finally, Section 6 summarizes our research and discusses future work.

2 Related Work

Almost every mainstream operating system has a single-host-version volume manager, for example, the dynamic disk manager in Windows [8], Logical Volume Manager (LVM) in Linux [9], and so on. Since these are inadequate for SAN environments where multiple hosts share storage devices, some volume managers supporting cluster environments have been designed, such as CLVM [10], the SANtopia volume manager [11], and Cluster EVMS [12]. They only apply to clusters running a single operating system, but, in fact, more than one operating system runs in most data centers.

IBM SAN Volume Controller [13] and DataCore SANSymphony [14] are typical in-band virtualization systems. Because all data are transported through the virtualization appliance, the latency of reads and writes increases. Moreover, the virtualization appliance is prone to becoming an I/O bottleneck of the whole system.

1. The chunk is the basic unit of data striping in striped volumes and its size is a multiple of the basic disk block size.

Out-of-band virtualization systems mainly include HP VersaStor [3], StoreAge SVM [4], V:drive [5], and SANfs-VM [6]. These systems are dependent on specific platforms and/or hardware. For instance, HP VersaStor only applies to specified host bus adapter (HBA) cards because its agent is implemented in the HBA firmware [3]. When V:drive or SANfs-VM is used, only Linux can run on the hosts.

Gonzalez and Cortes [15] proposed an algorithm for scaling RAID5 volumes which has an easily controlled overhead. The algorithm enables the newly added disks to be gradually available to serve user requests during the scaling process. As far as online redistribution of data on striped volumes is concerned, Ghandeharizadeh and Kim [16] proposed that a system should employ both lazy and eager reorganizations: Lazy reorganization is used to minimize the amount of wasted disk bandwidth by reorganizing the blocks that are staged in memory by a media display; eager reorganization prevents the system from wasting the idle disk bandwidth by employing it to reorganize files that are not referenced by a media display. Among all of the previous systems, only the SANtopia volume manager [11] and TH-VSS [7] can scale a striped volume online. However, they manage mapping information with a mapping table that expands with the volume size. This restricts their scalability.

The TH-VSS system [7] employs a mirroring mechanism to ensure that the data on striped volumes is not corrupted in the scaling process. However, it requires all of the disks in the original striped volume to have a large unused storage space. Ordered operations of copying a data chunk and updating the mapping information [17] can ensure the consistency of striping. However, this mechanism requires one write of mapping information for each data chunk movement, which results in a large cost of data redistribution. In [18], we presented the reordering window characteristic in the scaling process and proposed the SLAS approach to scale striped volumes in a disk array. The difference in the architecture between a disk array and an out-of-band virtualization system makes SLAS unable to work in MagicStore. With the help of the reordering window characteristic, we propose the SLAS2 approach to scale striped volumes efficiently in an out-of-band virtualization system. The detailed difference between SLAS and SLAS2 is discussed in Section 3.

3 MagicStore Design

3.1 System Architecture

MagicStore is made up of the manager on the virtualization appliance and the agent software on each client (Fig. 1). The manager knows the states of physical devices and manages virtualization metadata. Instructed by the manager, the agent creates logical volume devices and performs address mapping from the logical address space to the physical address space. Each agent is connected to the manager over an Ethernet network.

The agent consists of a mapper module, a batman thread in the kernel space, a loadconf utility, and a configuration file in the user space. The mapper is a lightweight driver

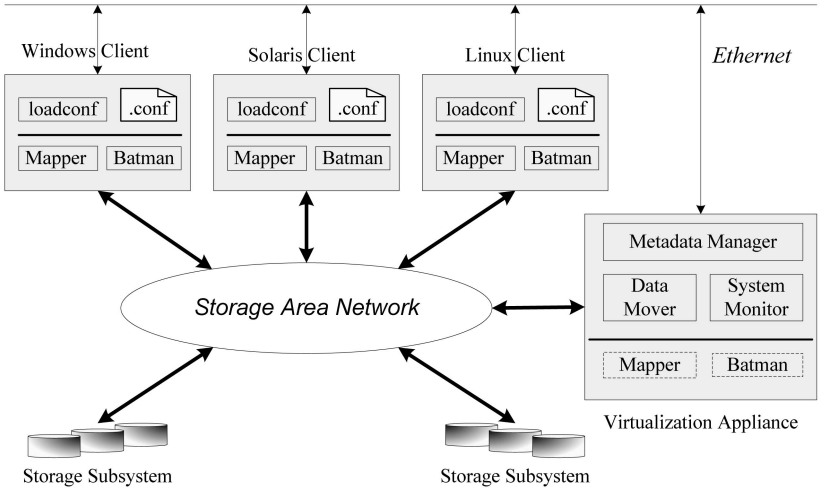

Fig. 1. Architecture of the MagicStore system.

that sits between the file system and the disk driver. It maps the I/O requests sent to logical volumes to the corresponding physical disks. When the mapper is loaded, it creates the batman, a kernel thread that receives and executes virtualization instructions from the manager. The batman might also be implemented in the user space, but frequent data copying between the kernel space and the user space will definitely impair system performance [19]. The configuration file mainly contains the information to communicate with the manager, such as the IP address and listening port of the manager. The loadconf utility is used to ask the mapper to reload the configuration information.

The manager consists of three cooperative modules: a metadata manager, a system monitor, and a data mover. The metadata manager manages virtualization metadata, sends virtualization instructions to agents, and processes mapping requests from agents. It organizes virtualization metadata with a three-layered model separating physical volumes, volume groups, and logical volumes [9]. Logical volumes may be allocated to hosts with access permissions. The system monitor collects the state information of the whole system dynamically. Only when the manager has accurate information about the state of a system can it perform virtualization management correctly. For instance, fabric zoning [20] may make a host unable to access some storage devices. Consequently, it is invalid for the manager to allocate to a host the storage resources on the devices unaccessible to it. The data mover moves data according to a given policy. For example, data redistribution for scaling a striped volume is performed by the data mover. In addition, the manager enables the applications on itself to access any logical volume by loading the agent software.

3.2 System Start-Up and I/O Processing

Fig. 2 shows how the MagicStore system starts up and how one I/O request from a client is processed by the system step by step. The manager scans physical disks to obtain the virtualization metadata on start-up. After booting, the agent on each client scans physical disks to get their disk identifiers (UUIDs), retrieves virtualization metadata from the manager, and creates virtual storage devices accordingly.

As for each logical volume, the mapper can use two modes to conduct address mapping: the cache mode and the noncache mode. By default, the mapper maps I/O requests in the cache mode. In this mode, the mapper holds the mapping information of a logical volume and calculates the address mapping for each I/O locally without communicating with the manager.

When a virtualization transaction such as online resizing requires modification of the metadata of a logical volume that has been allocated to a client, the manager first revokes

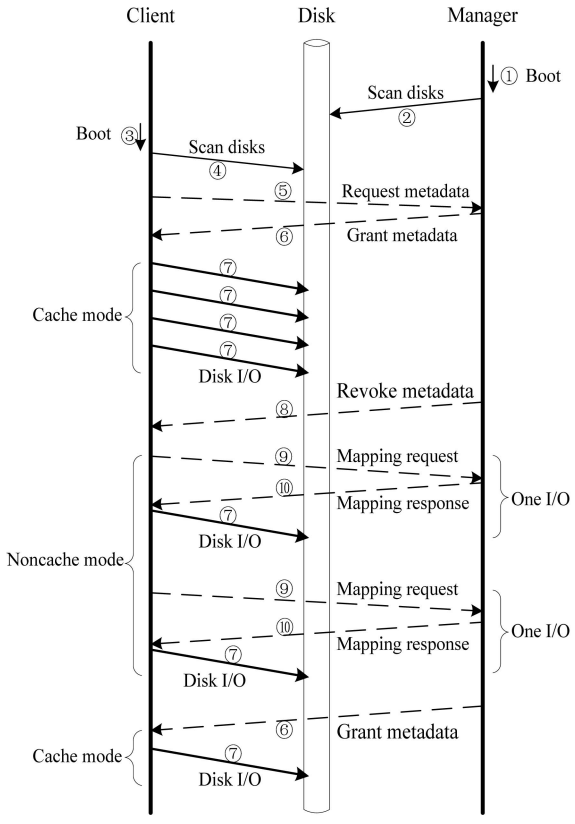

Fig. 2. Workflow of system start-up and I/O processing.

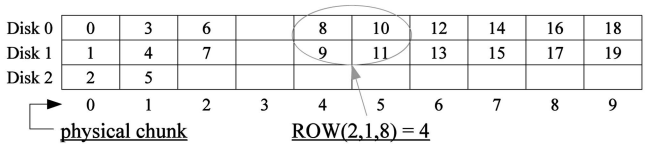

Fig. 3. A state of data redistribution (Disk 2 is newly added).

the metadata from the client to make the logical volume work in the noncache mode. In this mode, the mapper sends a mapping request to the manager for each I/O. After the transaction is finished, the manager grants the new metadata to the client and makes the logical volume switch back to the cache mode.

This caching mechanism of mapping metadata brings higher performance due to the reduction of network communication and provides better scalability since most I/O requests do not disturb the manager. In most cases, mapping metadata is not modified and MagicStore works in the cache mode. This can eliminate the overhead of network communication because no mapping request is sent to the manager. In this situation, whether the metadata server is implemented in pure software or using a hardware appliance makes no difference. Even if there is a performance difference between hardware appliances and MagicStore, the difference takes effect only when the mapping metadata is modified. Moreover, when the mapping metadata of a logical volume is modified, the other volumes can still be mapped in the cache mode.

The caching mechanism of mapping metadata also decreases the possibility that the manager becomes a single point of failures. In most cases, all of the clients map logical volumes in the cache mode. Therefore, they can still work well even when the manager fails. One exception is that some clients are mapping logical volumes in the noncache mode when the manager fails. One solution is to use dual managers in either load-balancing or failover mode, as many appliance vendors have done [1].

3.3 SLAS2 Approach to Scaling Striped Volumes

Solving the problem of scaling round-robin striped volumes can be accomplished by finding an efficient approach to redistributing the data. In addition to reads and writes of data chunks, data redistribution requires updates of mapping metadata to guarantee data consistency. Our focus is to improve the efficiency of data redistribution from two aspects: 1) reads and writes of data chunks and 2) updates of mapping metadata.

We found that, during the data redistribution process, there always exists a reordering window where consistency can be maintained while changing the order of data movements. Given a request to add m disks to a striped volume made up of n disks, if the size of the reordering window at Chunk x is represented as ROW(n, m, x), we proved that $\mathrm{ROW}(\mathrm{n},\mathrm{m},\mathrm{x}) = m \times \lfloor \mathrm{x}/\mathrm{n} \rfloor$ [18]. Taking a state of data redistribution for adding one disk into a two-disk striped volume (Fig. 3) as an example, when Chunks 8, 9, 10, and 11 are moved in an arbitrary order, no valid chunk will be overwritten. If Chunk 12 is also taken into account, when Chunk 12 is moved before Chunk 8, the former will overwrite the latter. Therefore, $\mathrm{ROW}(2,1,8) = 4$.

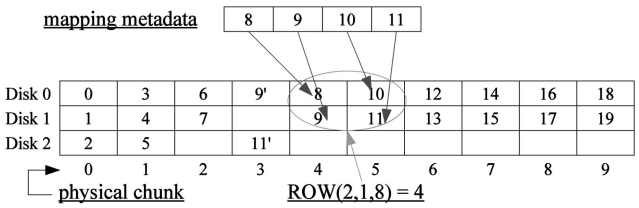

Fig. 4. If data chunks are copied to their new locations and metadata is not yet updated when the system crashes, data consistency is still maintained (the reason is that the data in their original locations are valid and available).

Further, suppose that the system fails when Chunks 9 and 11 have been copied to their new locations and the mapping metadata has not been updated (see Fig. 4). Their original replicas will be used after the system reboots. As long as Chunks 9 and 11 have not been written since they were copied, the data remains consistent.

The reordering window characteristic provides a theoretical underpinning for solving the problem of scaling striped volumes in an out-of-band virtualization system. The following conclusions can be drawn from the concept of a reordering window:

- If and only if data chunks coexist in the same reordering window is the order of their movements changeable.
- For almost every data chunk, even if mapping information is not updated immediately after it is copied to its new location, data consistency is still guaranteed.
- A data chunk may have two valid replicas when the chunk has been copied to its new location and has not been written since it was copied.

Taking advantage of the above conclusions, we propose SLAS2 (to be differentiated from SLAS in [18]), an efficient approach to scaling round-robin striped volumes in an out-of-band virtualization system. Before a striped volume is scaled, the mapping mode of the volume is switched to the noncache mode. The data mover in the manager performs data redistribution with the SLAS2 approach, which is described as follows:

- The data mover maintains a sliding window locally as mapping metadata, which occupies a very small space.
- The data mover uses lazy updates of mapping metadata to decrease the number of metadata writes required by data redistribution.
- The data mover also changes the order of data chunk movements so as to access multiple successive chunks via a single I/O.
- If a data chunk has two valid replicas that exist on two disks, read requests to the chunk are alternated between the two disks.

The traditional mapping-management solutions, that is, mapping function [9], [10] and mapping table [6], [11], either do not support the online redistribution of data or are inadequate for large-scale storage environments. SLAS2 uses a new mapping management solution based on a

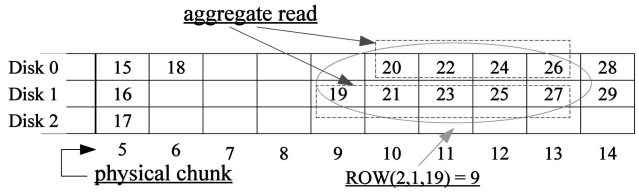

Fig. 5. Aggregate reads (multiple successive chunks are read via an I/O).

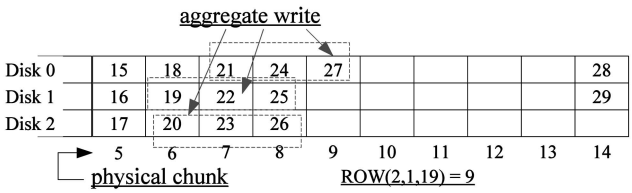

Fig. 6. Aggregate writes (multiple successive chunks are written via an I/O).

sliding window [17]. The key idea is to introduce the concept of a sliding window into the mapping function. Like a small mapping table, a sliding window describes the mapping information of a continuous segment of the striped volume. During data redistribution, only the data within the range of the sliding window are redistributed.

The data mover does not update the sliding window in memory onto a local disk until it has to do so to keep consistency. Data chunks are copied to new locations in some order, but writes of mapping metadata are done only under one of two circumstances: 1) When all of the data chunks in the sliding window have been copied to new locations, the window offset and the window size on the disk are updated and 2) when the first write to a data chunk in the sliding window arrives after the chunk is copied to its new location, the sliding window is updated and then the write request is served.

The data mover changes the movement order of data chunks in a sliding window so as to aggregate reads/writes of multiple data chunks. Taking the data redistribution state shown in Fig. 5 as an example, we get $\mathrm{ROW}(2,1,19) = 9$. The data mover issues an I/O request to read Chunks 20, 22, 24, and 26, and another one to read Chunks 19, 21, 23, 25, and 27. Thus, two instead of nine I/Os are required to read these chunks. When all of these chunks have been read into a memory buffer, the data mover issues a first I/O request to write Chunks 21, 24, and 27, a second one to write Chunks 19, 22, and 25, and a third one for Chunks 20, 23, and 26 (see Fig. 6). In this way, only three instead of nine write requests are issued.

A chunk in the sliding window has two valid replicas if it has been copied to its new location and has not been written since it was copied. If the two replicas do not exist on the same disk, read requests to the chunk are alternated between the two disks.

Although the SLAS approach working in a disk array [18] shares the same theoretical underpinnings with SLAS2, there are some technical differences between them which are summarized as follows:

- The architecture of a disk array is in-band in nature, so the data mover can make use of the content of the data accessed by user requests. Conversely, MagicStore exploits the out-of-band architecture. Hence, the data mover residing in the metadata server has no chance to get the accessed data. The difference in the architecture makes SLAS unable to work in MagicStore.
- SLAS stores the sliding window in a component disk of the striped volume, whereas SLAS2 stores it in a local disk of the manager. Metadata updates in SLAS cause a disturbance to the sequence of disk seeks required by user data accesses. Differently, SLAS2 updates mapping metadata on a local disk of the manager, which thoroughly eliminates the disturbance.
- When the data written by a user I/O overlaps with the data that has been read into memory and has not been copied to new locations by the data mover, SLAS and SLAS2 have different processing steps. In SLAS, the I/O processor can update the in-memory data piggyback, so the I/O aggregation is free of any negative impact. In SLAS2, however, because the data mover cannot get the update data, the related chunk is written to the new location before the mapping information is returned to the agent. Thus, the early write breaks the aggregated large write into multiple pieces.
- The in-memory data for I/O aggregation in SLAS can be used to serve read requests, whereas the in-memory data in SLAS2 cannot be seen by user I/Os at all.

The SLAS2 approach has some important advantages. First, a sliding window occupies a very small space, so SLAS2 provides better scalability. Second, one write of the sliding window can commit multiple mapping changes of data chunks, so lazy updates can decrease the number of metadata writes effectively. Third, I/O aggregation enables SLAS2 to have a larger redistribution throughput due to the decrease of disk seeks. Finally, alternating read requests to a chunk in the sliding window between two disks can shorten user I/O latencies.

3.4 Metadata Integrity Maintenance

To enable an out-of-band virtualization system to survive panics and power failures, it is necessary to maintain the integrity and consistency of virtualization metadata. When modifying the virtualization metadata of a logical volume, the manager revokes the metadata from the related client. Therefore, only the metadata integrity on the manager side needs to be ensured. MagicStore uses a combination of lazy synchronizations, ordered writes, and REDO logging to achieve this goal.

Each physical volume in a volume group holds a replica of the metadata describing the volume group. Storing the metadata on physical disks can bring more benefits than storing it on the manager locally. It bundles physical disks and logical volumes they describe together and uses multiple metadata replicas in each volume group to enhance the metadata reliability. In order to reduce the

overhead of metadata updates, MagicStore uses a lazy synchronization scheme to update multiple replicas in a volume group. The manager chooses two replicas as primary databases from those with the highest version numbers and accesses only these two replicas during virtualization transactions. Any other replica in the group is synchronized when the manager is not busy. Two replicas are chosen as primary databases in order not to leave a single point of failures.

When a virtualization transaction, for example, data migration for load balancing, requires movement of a physical chunk, the consistency between the virtualization metadata and the underlying physical data has to be maintained. The manager uses the method of ordered writes to guarantee consistency. Here, the physical chunk is first copied to its new location and the mapping information is then written to the disk. Even if the power fails in between, the physical chunk in the original location will be used because the mapping information has not been updated. Nevertheless, the opposite order is problematic.

Some virtualization transactions write multiple metadata blocks. For example, creating a logical volume will write both the volume group entry and the logical volume entry. MagicStore uses REDO logging to ensure that writes to multiple metadata blocks in a single virtualization transaction are atomic. In this case, the operation is logged before the metadata can be updated. In the event of a crash, it scans through and replays the log when the manager reboots. Thus, the metadata remains consistent.

3.5 Incorporating Legacy Storage

In typical legacy storage systems, a physical disk is divided into several volumes through the partitioning technique and each volume stores a bulk of data. To make an out-of-band virtualization system incorporate the storage resources in a legacy storage system, the traditional method creates a logical volume for each legacy volume and then copies the entire content of the legacy volume. As for a large data center like the NCCS, copying the data is quite time consuming [21] and even makes the application services unavailable during the process.

Since the traditional method is expensive, we propose an effective method for incorporating typical legacy storage, which is described in Fig. 7. The amount of the data moved by our method is equal to the size of the metadata database and is independent of the capacity of the disk. We let s denote the size of the metadata database. If there are k legacy disks to incorporate, then only $s \times k$ data needs to be moved. Therefore, our new method can incorporate a large legacy storage system quickly.

If a legacy disk returns to the legacy system, only the data on the space S_i needs to be moved to the reserved location RL_i (see Fig. 7). However, if the traditional method is used, this also needs to move all of the data on the disk.

Step 1. add a disk *ND* and initialize it into a physical volume

Step 2. if all legacy disks have been incorporated, exit; else, select an unincorporated legacy disk LD_i

Step 3. obtain the volume layout on the disk LD_i by analyzing the partition table

Step 4. move the data on the reserved location RL_i of LD_i to the unused space S_i of *ND*

Step 5. make the disk LD_i a physical volume PV_i by writing metadata on the location RL_i

Step 6. reconstruct the volume(s) on the disk LD_i by mapping the valid space of PV_i and the space S_i

Step 7. GOTO *Step 2*.

Note that the reserved location RL_i is reserved for the metadata database in the MagicStore system.

Fig. 7. New method for incorporating typical legacy storage.

4 MagicStore Implementation

In this section, we first present how client agents on different platforms are implemented. Then, we give the implementation details of the metadata server, including our method for incorporating legacy storage. Finally, we describe how the SLAS2 approach is implemented in the out-of-band architecture.

4.1 Client Agent Implementation

In Windows 2000/XP/2003 [8], the DriverEntry routine of the mapper driver initializes the batman thread, which creates a disk device object to represent each logical volume. When an I/O request packet (IRP) arrives, a dispatch routine maps it to the physical address space and allocates several derived IRPs. The routine then registers a completion routine for each derived IRP and, finally, sends the derived IRP to the physical disk driver.

In Solaris 10 [22], the mapper driver first creates the batman thread in the attach routine. To represent a logical volume, the batman creates a minor device. When a logical volume is accessed, the strategy routine maps the logical address to the physical address. Then, it clones derived buf(s) and registers a completion routine for each one. Finally, the strategy routine sends each derived buf to the physical disk driver.

The implementations in Linux 2.4 and Linux 2.6 [23] are similar. The only difference is that their units of address mapping are a bh and a bio, respectively. In Linux 2.4, for example, the request routine in the mapper driver performs address mapping and rewrites the b_rdev field to specify a physical volume. Then, it registers a completion routine for the bh and returns a positive value. Here, the generic_make_request routine chooses the physical disk driver according to the new major number.

The manager and the agents communicate via a lightweight UDP. The batman creates a UDP server in the kernel space to watch for virtualization instructions from the manager. Likewise, the mapper creates a UDP client and sends a mapping request to the manager for each incoming I/O when the target volume works in the noncache mapping mode.

4.2 Metadata Server Implementation

The manager is implemented in the user space on the Linux platform. It issues I/Os to the disks by accessing block

TABLE 1
Configuration of the Machines

	CPU	Memory	FC HBA	OS	File System
Windows Client	Intel Xeon 2.4 GHz×2	1 GB	Emulex LP982	Windows Server 2003	NTFS
Solaris Client	UltraSPARC-IIi 300 MHz×2	256 MB	Emulex LP9802	Solaris 10 64-bit	UFS
Linux 2.4 Client	Intel Xeon 2.4 GHz×2	1 GB	Emulex LP982	Linux 2.4.26	EXT2
Linux 2.6 Client	Intel Xeon 2.4 GHz×2	1 GB	Emulex LP982	Linux 2.6.10	EXT2
Metadata Server	Intel Xeon 2.4 GHz×2	1 GB	Emulex LP982	Linux 2.4.26	EXT2

devices directly. MagicStore provides three mapping schemes from logical volumes to physical volumes: linear mapping, striped mapping, and mirrored mapping. To enable the manager and clients to recognize a physical volume uniquely, the metadata manager writes a label and a UUID onto the reserved location of each target device. The metadata manager puts a physical volume into an appropriate volume group according to its property in bandwidth, latency, and reliability.

To incorporate a legacy disk, the metadata manager first reads the master boot record on its first sector to retrieve the partition information. It also reads the information of extended partitions, if any. Then, the metadata manager moves the data from the location where the metadata database will be placed to the space allocated on the physical volume newly added for the incorporation. Finally, the metadata manager initializes the legacy disk into a physical volume by writing virtualization metadata onto it, puts the two physical volumes into a volume group, and reconstructs a logical volume for each partition according to the partition table.

To achieve centralized monitoring and provide good flexibility, the system monitor uses a two-layered architecture, including an object manager and three providers. The three providers get state information of storage devices, fabric links, and hosts through an HBA API, a switch API, and a host interface, respectively. All state information will be provided to the object manager as uniform objects. The object manager monitors the occurrence of events and performs some actions.

4.3 SLAS2 Implementation

On scaling a striped volume, the metadata manager initializes a read/write lock for each entry in the sliding window. Then, it creates the data mover thread to redistribute the data on the volume.

When an I/O mapping request arrives, the metadata manager performs address mapping and returns the results to the client directly if the I/O does not overlap with the current sliding window. Otherwise, the metadata manager locks a new mutex and creates a logical I/O thread. After acquiring related read-write lock(s) for reading, the logical I/O thread returns the mapping result to the client and then locks the previous mutex. Because the mutex has already been locked by the metadata manager, the thread blocks until the mutex becomes available. After an I/O is completed, the mapper on the client tells the metadata manager to unlock the mutex if a logical I/O thread has acquired related read-write lock(s) for the I/O. Since the mutex is available now, the logical I/O thread stops blocking and unlocks related read-write lock(s).

The data mover thread stores the metadata in the local file during the scaling process and uses lazy updates of mapping metadata to decrease the number of metadata writes. To perform I/O aggregation, the data mover first takes two steps for each disk in the original volume: 1) Given the size of the reserved memory for I/O aggregation, it calculates the size of successively readable data on the disk and 2) it acquires related read-write lock(s) for writing, reads all of the data via a single I/O, and, finally, unlocks related read-write lock(s). After all of the data that the reserved memory can hold are read, the data mover takes another two steps for each disk in the new volume: 1) Given the size of the reserved memory, it calculates the size of successively writable data on the disk and 2) it acquires related read-write lock(s) for writing, picks up the data that will be written to the disk from the reserved memory into the write buffer, writes all of the data via a single I/O, and, finally, unlocks related read-write lock(s).

5 Experimental Evaluation

We first measured the performance of client agents in the cache mode by comparing the performance of logical volumes managed by MagicStore with that of plain volumes managed by the original operating systems. Second, we demonstrated the performance difference between the cache mode and the noncache mode. Third, we examined the scalability of the metadata server implemented in pure software. Finally, we evaluated the features of MagicStore, including scaling striped volumes and incorporating legacy storage. Although a set of performance evaluation results based on a comparison with hardware out-of-band virtualization systems could be more convincing, we did not do so because it was very difficult for us to obtain such a hardware virtualization system.

The testbed used in these experiments is described as follows: The agents for different platforms and the manager were installed on some machines whose configurations are shown in Table 1. Via a Brocade Silk Worm 3800 FC switch, these machines were connected with an FC disk array controlling several Seagate ST3146807FC disks. All of these machines were also connected via a 100 Mbps Ethernet. We used Iometer [24] to generate representative workloads. Iometer can generate different workloads of various characteristics, including the read/write ratio, the request size, and the maximum number of outstanding requests.

5.1 Performance of Client Agents

We began our performance evaluation of MagicStore with measuring the performance of its client agents in the cache mode. We expected that the client agents would add little overhead despite performing extra address mapping for each I/O. On different platforms, we tested the performance of plain volumes, linear volumes, and striped volumes with a file system mounted. We configured Iometer to generate the workloads that used random addresses with the transfer request size doubled from 8 Kbytes to 4 Mbytes. All of the workloads consisted of 20 percent writes and 80 percent reads since Vogels found that 79 percent of accesses to files were read-only [25].

The bar graph in Fig. 8 plots the throughput of linear volumes and plain volumes on the Windows, Linux 2.4, and Solaris platforms as the request size increases along the x-axis. We find that the performance of linear volumes is quite close to that of plain volumes on all of the platforms. On average, the performance declines introduced by linear volumes were 0.3 percent on the Windows platform, 2.1 percent on the Linux 2.4 platform, and 0.4 percent on the Solaris platform.

The line and symbol graph in Fig. 8 plots the comparison in CPU utilization between linear volumes and plain volumes. It shows that, while performing address mapping for linear volumes, the client agents introduced a very small increase of CPU utilization, about 0.017 percent, 0.116 percent, and 0.777 percent, respectively, on the Windows, Linux 2.4, and Solaris platforms.

As the figure shows, the CPU utilization on the three platforms varies greatly. First, the CPU utilization on the Linux 2.4 machine is much higher than that on the Windows machine. This unexpected phenomenon is due to one of the design pitfalls in the Linux 2.4 kernel: The buffer head forces the kernel to break up potentially large block I/O operations into many multiple bh structures [23]. This pitfall causes the client agent to perform address mapping much more frequently. Due to the limitations of the bh structure, the Linux 2.6 kernel uses the bio structure as the basic container for a block I/O. As the latter experimental results show, the CPU utilization of the client agent on the Linux 2.6 machine is much lower. Second, the CPU utilization on the Solaris platform increases with the transfer request size sensitively. The main reason is that the 300 MHz CPU had to perform data copying between the user space and the kernel space. As the throughput increased with the transfer request size, the CPU utilization became higher. However, this did not reflect the current typical machine configurations. We will redo the experiments once a Sparc machine with a higher configuration is available.

All of the striped volumes in this set of experiments were constructed of four disks with a stripe size of 32 Kbytes or 64 Kbytes. As shown in the bar graph in Fig. 9, although there was a clear benefit in using striped volumes for both the stripe sizes, the benefit was more apparent for the 64 Kbyte stripe size. In the case of 32 Kbytes, we achieved around 88 percent performance improvement on the Windows platform; the improvement on the Linux 2.4 platform was about 25 percent. When the stripe size was

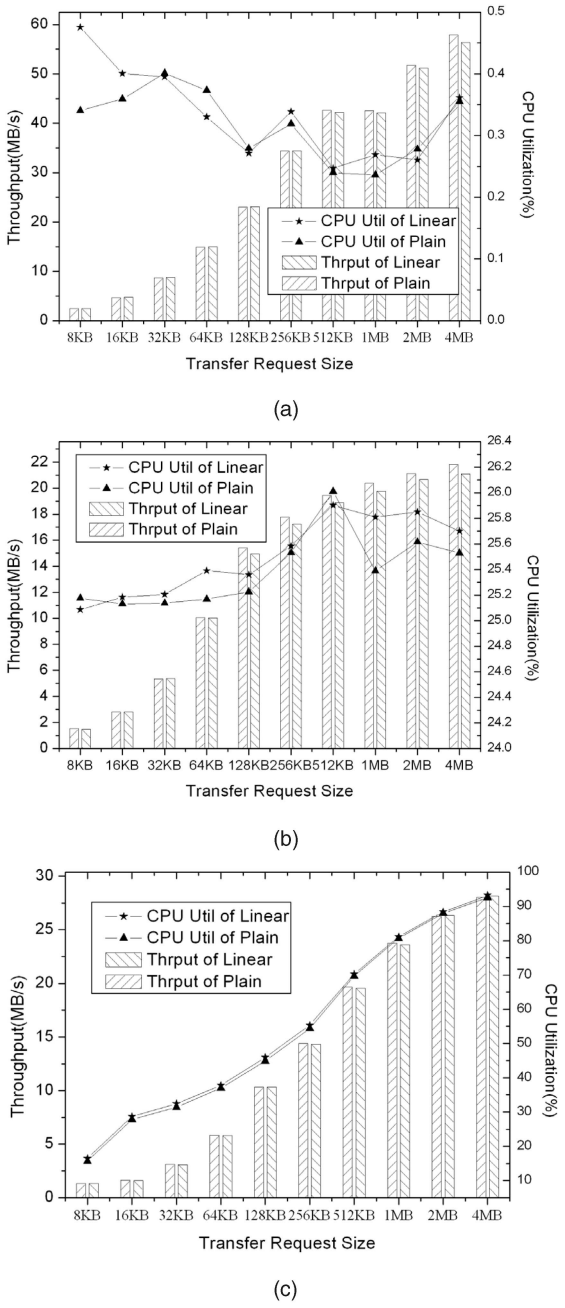

Fig. 8. Performance comparison between linear and plain volumes. (a) On the Windows platform. (b) On the Linux platform. (c) On the Solaris platform.

64 Kbytes, performance was improved by 104 percent and 30 percent, respectively. This indicates that the stripe size has an effect on the performance of a striped volume.

Due to its more complex algorithm, we imagined that mapping for striped volumes would impose a higher processing overhead than mapping for the previous linear volumes. The line and symbol graph in Fig. 9 shows a comparison of CPU utilization for striped volumes and plain volumes on the Windows and Linux 2.4 platforms. In the case of 32 Kbytes, striped volumes introduced 1.310 percent and 0.278 percent CPU utilization increases on the Windows and Linux 2.4 platforms, respectively. When the stripe size was 64 Kbytes, the CPU utilization increases on

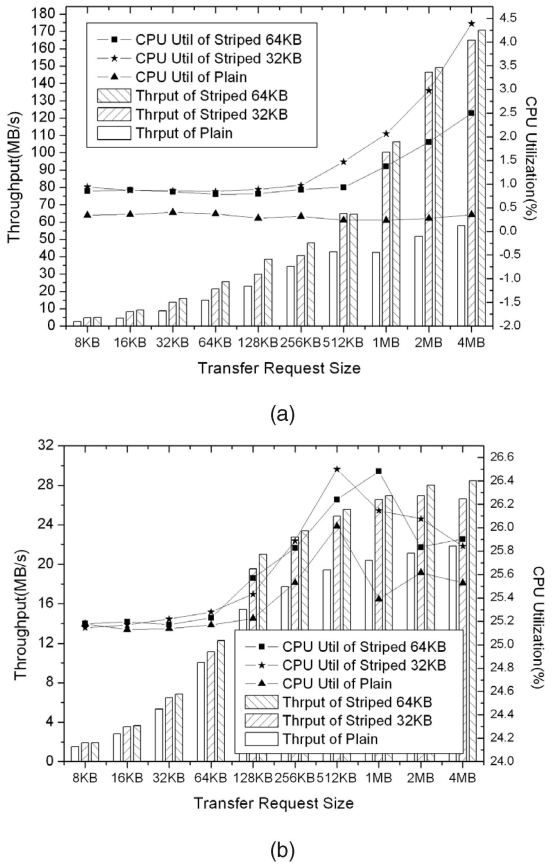

Fig. 9. Performance comparison between striped and plain volumes. (a) On the Windows platform. (b) On the Linux platform.

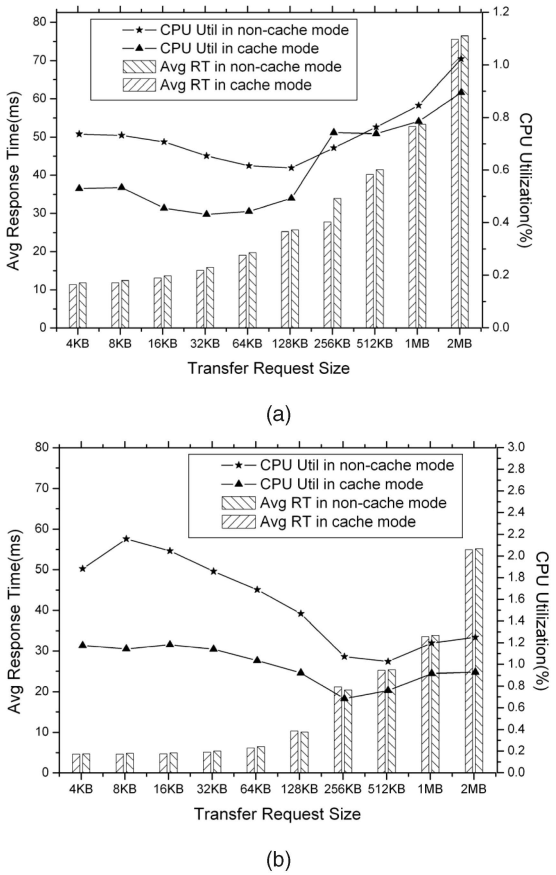

Fig. 10. Performance comparison between the cache and noncache modes. (a) Random workload. (b) Sequential workload.

the two platforms were about 0.855 percent and 0.272 percent, respectively. This indicates that the increase in CPU utilization introduced by the client agents performing address mapping for striped volumes was negligible. Our results also show that the CPU utilization increase for the stripe size of 64 Kbytes was less than that for the stripe size of 32 Kbytes. This is caused by different request sizes for each individual disk due to different stripe sizes.

Finally, we can observe a fluctuation of CPU utilization curves with a peak at the request size of 512 Kbytes and 1 Mbyte on the Linux 2.4 machine. This may be because the buffer head in Linux 2.4 forces the kernel to perform a splitting-merging operation for each I/O. The increase in the request size has two impacts: First, it results in a higher operation complexity and, second, it decreases the operation frequency. When the request size is relatively small, the first impact dominates, hence causing a higher CPU utilization. After the request size reaches a critical point (that is, 512 Kbytes or 1 Mbyte), the second impact dominates and leads to a decrease in CPU utilization.

5.2 Performance Comparison between the Cache and Noncache Modes

We use the caching mechanism of mapping metadata to improve the performance and scalability of the system, so it is interesting to see the actual performance difference between the cache mode and the noncache mode. We used four Iometer workers running on the Linux 2.6 machine to generate random and sequential workloads, with the request size varying from 4 Kbytes to 2 Mbytes. The target volume was a striped volume across four disks. We collected the response time of I/O requests and sampled the CPU utilization of the client.

Fig. 10 gives a performance comparison between the two mapping modes. Compared with the noncache mode, the cache mode brought a reduction of 1.236 ms in average response time and a decline of 0.13 percent in CPU utilization under the random workload; it also provided a reduction of 0.084 ms in average response time and a decline of 0.58 percent in CPU utilization under the sequential workload. Here, the cache mode brought a small performance enhancement because the manager served only one client. In a large SAN environment, the improvement brought by the caching mechanism will be more obvious. Furthermore, in the cache mode, the metadata server is not involved in any I/O. Consequently, the caching mechanism is helpful for enhancing the system scalability.

5.3 Scalability of the Metadata Server

To demonstrate the scalability of the metadata server implemented in pure software, we increased the number of clients that communicated with the metadata server to complete address mapping for each I/O. Each client ran on Linux 2.6 and had a striped volume across two disks. The Iometer running on each client used one or four workers to generate random or sequential workloads with 4 Kbyte and

64 Kbyte request sizes. We collected the round-trip time of mapping requests in the kernel space of clients and sampled the CPU utilization of the manager software. Fig. 11 shows how the performance of the metadata server is affected by the increase in the client number.

As far as each situation is concerned, it is shown that, with the increase in the client number, the CPU utilization of the manager software increased, whereas the average round-trip time of mapping requests remained 300 μs or so. The results indicate that, in the same situation, the CPU utilization under the sequential workload is markedly higher than that under the random workload. This is because more pressure is put on the manager due to the better performance of sequential accesses to physical disks. The results also show that the CPU utilization with the 4 Kbyte request size is obviously higher than that with the 64 Kbyte request size. This is because a small I/O size leads to an increased total number of requests, which, in turn, causes a higher mapping overhead.

When four clients used one worker to generate sequential workload with the 4 Kbyte request size, respectively, the CPU utilization reached the maximum 2.4 percent. Therefore, the metadata server has good scalability and the MagicStore system is adequate for large SANs with the help of the caching mechanism of mapping metadata.

5.4 Scaling Round-Robin Striped Volumes

The purpose of this group of experiments is to quantitatively characterize the overhead of online redistribution with the SLAS2 approach. We added one disk into a striped volume across three disks and used one or four Iometer workers to generate random workloads, with the request size varying from 4 Kbytes to 2 Mbytes. In all of these experiments, the sliding window sizes were set to 1,024 and the reserved memory for I/O aggregation could hold 40 chunks. We collected the response time of user I/Os on Linux 2.6 during the scaling process and the time for performing data redistribution. To provide a comparison, we also measured the response time of user I/Os sent to a four-disk striped volume working in the noncache mode without scaling.

Fig. 12 shows the comparison in average response time without and during the scaling and also gives the data redistribution time. When one Iometer worker was used, the online redistribution increased I/O latency by 3.26 ms; when four workers were used, the increase was 6.65 ms. As the I/O size increased, the redistribution time obviously increased. When one Iometer worker was used, the redistribution time varied from 344 sec to 778 sec; when four workers were used, it increased from 449 sec to 998 sec. Though not shown in the figure, we should point out that the redistribution time became remarkably longer when the reserved memory for I/O aggregation shrank to holding only four chunks. Due to the reduction of metadata updates and the aggregation of data accesses, SLAS2 can perform data redistribution in a shorter time with small foreground I/O latencies.

5.5 Incorporating Legacy Storage

To evaluate the performance benefits of our method for incorporating legacy storage, we measured the time for the manager to incorporate a legacy volume whose size ranged

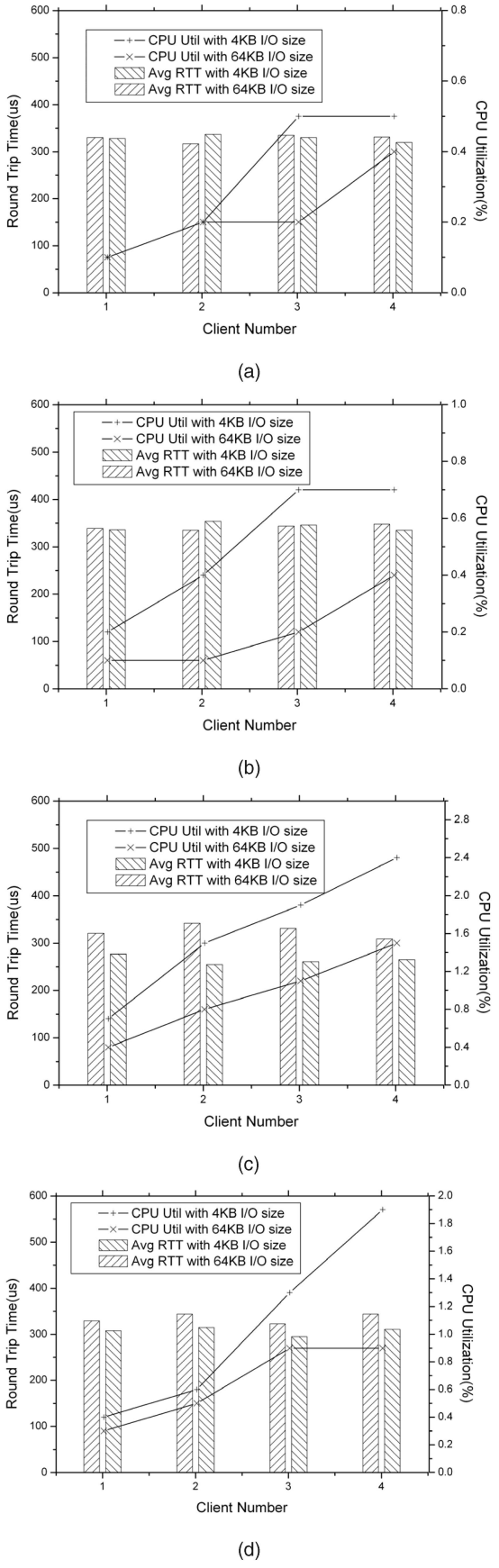

Fig. 11. Scalability of the metadata server. (a) Random workload and one worker. (b) Random workload and four workers. (c) Sequential workload and one worker. (d) Sequential workload and four workers.

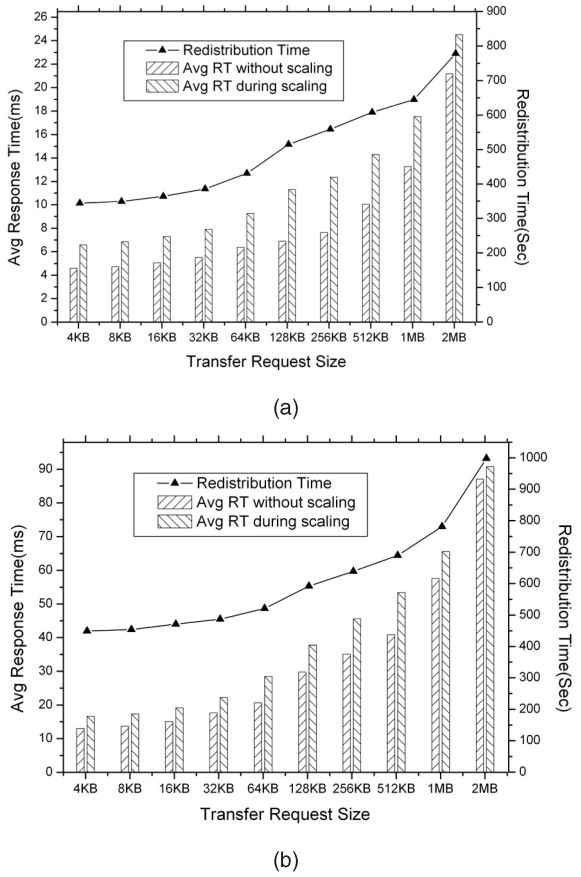

Fig. 12. Performance of the SLAS2 approach. (a) One test worker. (b) Four test workers.

from 10 Gbytes to 100 Gbytes with the traditional method and our method. As for the traditional method, which copies the entire content of the legacy volume, we neglected the time for creating a logical volume and only collected the time for copying the entire content.

Table 2 compares the performance of incorporating legacy storage with the two methods. As shown in the table, the traditional method required linearly increasing time to incorporate a legacy volume as the volume size increased. For a 100 Gbyte legacy volume, the traditional method even required 3,337 sec (about 56 minutes). Conversely, our method required a very short time and stabilized between 337 ms and 608 ms. Suppose that a data center incorporates a mass of legacy storage (for example, 100 146 Gbyte legacy disks), the traditional method will be badly time-consuming and will result in an unacceptable downtime, whereas our method will complete the incorporation quickly.

6 Conclusions and Future Work

In this paper, we present and evaluate the design and implementation of an out-of-band virtualization system, MagicStore, which is implemented through pure software and supports three platforms: Windows, Solaris, and Linux. Some conclusions about MagicStore can be drawn from our experimental results:

- The client agents add little overhead despite performing address mapping for each I/O.
- The metadata server has good scalability, and MagicStore is adequate for large SANs with the help of the caching mechanism of mapping metadata.
- MagicStore can complete data redistribution required by scaling a striped volume in a short time with small user I/O latencies.
- MagicStore can incorporate large legacy storage quickly.

In the out-of-band architecture, the absence of a virtualization appliance in the data access path also presents a security question: How do we prevent a client from accessing the blocks it does not own? Because the appliance has no chance to do verification, storage devices have to execute some form of cryptography or authentication on every I/O. For example, the capability-based methods [26], [27] require intelligent disks with a powerful processing capability and an evolved Small Computer System Interface (SCSI) command set. Since this requirement exceeds the capabilities of modern disk drives, we have not implemented access authentication in the current MagicStore. When first-generation intelligent disks are available, we will develop a method for managing access authentication in MagicStore. Our preliminary idea is that the client asks the manager for a ticket vouching for the client's ability to access certain data on physical disks. The client passes such a ticket to the disk, which verifies the manager's approval.

Acknowledgments

The authors would like to thank the anonymous reviewers for their suggestions on improving this paper. This work was supported by the National Natural Science Foundation of China under Grants 60433040 and 10576018, the National Grand Fundamental Research 973 Program of China under Grant 2004CB318205, and the Program for New Century Excellent Talents in University.

TABLE 2
Incorporation Time Comparison between the Traditional Method and Our Method

Volume size	10GB	20GB	30GB	40GB	50GB	60GB	70GB	80GB	90GB	100GB
Traditional method	320s	621s	945s	1269s	1581s	1911s	2271s	2591s	2947s	3337s
Our method	349ms	531ms	453ms	608ms	337ms	407ms	416ms	395ms	401ms	577ms

REFERENCES

[1] T. Clark, *Storage Virtualization: Technologies for Simplifying Data Storage and Management.* Addison-Wesley Professional, Mar. 2005.

[2] R. Barker and P. Massiglia, *Storage Area Network Essentials: A Complete Guide to Understanding and Implementing SANs,* C.A. Long, ed., p. 346, John Wiley & Sons, Oct. 2001.

[3] S. Sturgeon and T. Anderson, *QLogic Adopts Compaq VersaStor Technology for Storage Virtualization.* QLogic Corp., http://www.qlogic.com/news-events/details/releases_details.asp?id=624, 2007.

[4] "SVM—Storage Virtualization Manager," eng. white paper, StoreAge Networking Technologies Corp., http://www. storeage.com/media/upload/Datasheet%20-%20SVM%20with%20Data%20Path%20Module.pdf, Sept. 2004.

[5] A. Brinkmann et al., "V:Drive—Costs and Benefits of an Out-of-Band Storage Virtualization System," *Proc. 12th NASA Goddard, 21st IEEE Conf. Mass Storage Systems and Technologies (MSST '04),* pp. 153-157, Apr. 2004.

[6] S.-H. Lim et al., "Resource Volume Management for Shared File System in SAN Environment," *Proc. 16th Int'l Conf. Parallel and Distributed Computing Systems (PDCS '03),* 2003.

[7] D. Xiao, J. Shu, W. Xue, and W. Zheng, "TH-VSS: An Asymmetric Storage Virtualization System for the SAN Environment," *Proc. Int'l Conf. Computational Science,* vol. 3, pp. 399-406, 2005.

[8] D.A. Solomon and M.E. Russinovich, *Inside Microsoft Windows 2000,* third ed. Microsoft Press, Aug. 2000.

[9] D. Teigland and H. Mauelshagen, "Volume Managers in Linux," *Proc. Usenix Ann. Technical Conf.,* pp. 185-198, June 2001.

[10] H. Mauelshagen, "Linux Cluster Logical Volume Manager," *Proc. 11th Int'l Linux System Technology Conf.,* Sept. 2004.

[11] C.-S. Kim, G.-B. Kim, and B.-J. Shin, "Volume Management in SAN Environment," *Proc. Eighth Int'l Conf. Parallel and Distributed Systems (ICPADS '01),* pp. 500-505, 2001.

[12] R. Pai, "EVMS Cluster Design Document Version 2.0," http://evms.sourceforge.net/clustering/, 2007.

[13] J.S. Glider, C.F. Fuente, and W.J. Scales, "The Software Architecture of a SAN Storage Control System," *IBM Systems J.,* vol. 42, no. 2, 2003.

[14] DataCore Software, "DataCore SANsymphony 6.0—The Perfect Complement to Virtualized Server Infrastructure," www.datacore. com/downloads/SANSymphony%206%200%20Product%20Profile%20-%20March%202007%20-%20Final.pdf, Mar. 2007.

[15] J.L. Gonzalez and T. Cortes, "Increasing the Capacity of RAID5 by Online Gradual Assimilation," *Proc. Int'l Workshop Storage Network Architecture and Parallel I/Os,* Sept. 2004.

[16] S. Ghandeharizadeh and D. Kim, "On-Line Reorganization of Data in Scalable Continuous Media Servers," *Proc. Seventh Int'l Conf. Database and Expert Systems Applications,* D.G. Feitelson and L. Rudolph, eds., pp. 751-768, 1996.

[17] G. Zhang, J. Shu, W. Xue, and W. Zheng, "MagicStore: A New Out-of-Band Virtualization System in SAN Environments," *Proc. IFIP Int'l Conf. Network and Parallel Computing (NPC '05),* pp. 379-386, Nov.-Dec. 2005.

[18] G. Zhang, J. Shu, W. Xue, and W. Zheng, "SLAS: An Efficient Approach to Scaling Round-Robin Striped Volumes," *ACM Trans. Storage,* vol. 3, no. 1, article 3, Mar. 2007.

[19] M. Vilayannur, R.B. Ross, P.H. Carns, R. Thakur, A. Sivasubramaniam, and M. Kandemir, "On the Performance of the POSIX I/O Interface to PVFS," *Proc. 12th Euromicro Conf. Parallel, Distributed, and Network-Based Processing (PDP '04),* p. 332, 2004.

[20] C. Beauchamp and J. Judd, *Building SANs with Brocade Fabric Switches,* C.B. Nolan and K. Glennon, eds., p. 208. Syngress Publishing, Inc., Jan. 2001.

[21] E. Salmon, A. Tarshish, S. Patel, M. Saletta, M. Rouch, R. Caine, J. Paffel, L. Burns, E. Vanderlan, N. Palm, and D. Duffy, "Hierarchical Storage Management at the NASA Center for Computational Sciences: From Unitree to SAM-QFS," *Proc. 12th NASA Goddard, 21st IEEE Conf. Mass Storage Systems and Technologies (MSST '04),* pp. 101-107, 2004.

[22] Sun Microsystems, Inc., Part No: 816-4854-10, "Writing Device Drivers," pp. 199-214-255-274,http://docs-pdf.sun.com/816-4854/816-4854.pdf, Jan. 2005.

[23] R. Love, *Linux Kernel Development,* Z. Brown ed., pp. 212-215. SAMS, Developer Library Series, Sept. 2003.

[24] Intel Corp, "Iometer," http://www.iometer.org, July 2004.

[25] W. Vogels, "File System Usage in Windows NT 4.0," *Proc. 17th ACM Symp. Operating Systems Principles,* pp. 93-109, Dec. 1999.

[26] G.A. Gibson, D.F. Nagle, K. Amiri, J. Butler, F.W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka, "Cost-Effective, High-Bandwidth Storage Architecture," *Proc. Eighth Conf. Architectural Support for Programming Languages and Operating Systems,* 1998.

[27] B.C. Reed, D.D.E. Long, E.G. Chron, and R.C. Burns, "Authenticating Network Attached Storage," *IEEE Micro,* vol. 20, no. 1, Jan./Feb. 2000.

Guangyan Zhang received the bachelor's and master's degrees in computer science from Jilin University in 2000 and 2003, respectively. He is now a doctoral candidate in the Department of Computer Science and Technology at Tsinghua University. His current research interests include mass storage, parallel file systems, computer networks, and distributed systems.

Jiwu Shu received the PhD degree in computer science from Nanjing University in 1998. In 2000, he finished his postdoctoral position research at Tsinghua University and has been teaching at Tsinghua University since then. He is now a professor in the Institute of High Performance Computing Technology, Department of Computer Science and Technology, Tsinghua University. His current research interests include storage area networks, parallel and distributed computing and networking, algorithm analysis and design and parallel processing techniques, and cluster systems and communication.

Wei Xue received the PhD degree in electrical engineering from Tsinghua University, China, in 2003. In 2003, he joined the faculty of the Department of Computer Science and Technology at Tsinghua University. His research interests include cluster computing and network storage.

Weimin Zheng received the master's degree from Tsinghua University in 1982. He is the director of the Institute of High Performance Computing Technology, Department of Computer Science and Technology, Tsinghua University, China. Since 1982, he has been working at Tsinghua University in the area of parallel and distributed processing. His research covers parallel computer architecture, parallel and distributed computing, AI-oriented computer architecture, compiler techniques, and runtime system design for parallel processing systems and grid computing and network storage.