SLAS: An Efficient Approach to Scaling Round-Robin Striped Volumes

GUANGYAN ZHANG, JIWU SHU, WEI XUE, and WEIMIN ZHENG
Tsinghua University

Round-robin striping, due to its uniform distribution and low-complexity computation, is widely used by applications which demand high bandwidth and massive storage. Because many systems are nonstoppable when their storage capacity and I/O bandwidth need increasing, an efficient and online mechanism to add more disks to striped volumes is very important. In this article, it is presented and proved that during data redistribution caused by scaling a round-robin striped volume, there is always a reordering window where data consistency can be maintained while changing the order of data movements. Furthermore, by exploiting the reordering window characteristic, SLAS is proposed to scale round-robin striped volumes, which reduces the cost of data redistribution effectively. First, SLAS applies a new mapping management solution based on a sliding window to support data redistribution without loss of scalability; second, it uses lazy updates of mapping metadata to decrease the number of metadata writes required by data redistribution; third, it changes the order of data chunk movements to aggregate reads/writes of data chunks. Our results from detailed simulations using real-system workloads show that, compared with the traditional approach, SLAS can reduce redistribution duration by up to 40.79% with similar maximum response time of foreground I/Os. Finally, our discussion indicates that the SLAS approach works for both disk addition and disk removal to/from striped volumes.

Categories and Subject Descriptors: H.3.2 [**Information Storage and Retrieval**]: Information Storage; H4.2 [**Information Systems Applications**]: Types of Systems

General Terms: Algorithms, Management

Additional Key Words and Phrases: Striped volume, online scaling, reordering window, sliding window, lazy updates, I/O aggregation

ACM Reference Format:

Zhang, G., Shu, J., Xue, W., and Zheng, W. 2007. SLAS: An efficient approach to scaling round-robin striped volumes. ACM Trans. Storage 3, 1, Article 3 (March 2007), 39 pages. DOI = 10.1145/1227835.1227838 http://doi.acm.org/ 10.1145/1227835.1227838

This research was supported by the National Natural Science Foundation of China under Grant Nos. 60433040 and 10576018; the National Grand Fundamental Research 973 Program of China under Grant No. 2004CB318205; and Program for New Century Excellent Talents in University.

Authors' addresses: Department of Computer Science and Technology, Tsinghua University, 100084 Beijing, P. R. China; email: zhang-gy04@mails.tsinghua.edu.cn, {shujw, xuewei,zwm-dcs}@tsinghua.edu.cn.

1. INTRODUCTION

1.1 Motivation

The increasing performance gap between disk and memory [Yu et al. 2000] causes disks to remain an I/O bottleneck of the whole computer system. Striping data across multiple disks can provide effective load balancing of small I/Os and parallel transfers for large I/Os and, therefore, bring significant performance benefits. However, one of the most important problems in current systems is that applications often demand increasingly higher I/O performance and larger storage capacity [Ghandeharizadeh and Kim 1996]. Striping can solve the problem in a very simple way by simply adding more disks to striped volumes. On the one hand, adding disks increases the bandwidth because more parallelism can be obtained. On the other hand, more disks means more capacity since the redundancy overhead is not raised by the addition of new disks.

There are two well-known striping policies, round-robin policy [Schindler et al. 2004; Chen and Patterson 1990] and random policy [Alemany and Thathachar 1997; Goel et al. 2002]. Round-robin policy stripes the data across the disk set in a round-robin fashion, while random policy disseminates the data across the disk set randomly. Random striping appears to be more flexible when adding new disks or deleting existing disks. Unfortunately, due to its poor performance and lack of qualified randomized hash function, random striping is not so satisfactory a solution as expected. Conversely, round-robin striping, because of its uniform distribution and low-complexity computation, is widely used by applications which demand high bandwidth and massive storage. A typical example is that round-robin striping can be done in most disk arrays, logical volume managers, and file systems. Because many systems are nonstoppable when their storage capacity and I/O bandwidth need increasing, an efficient and online mechanism to add more disks to round-robin striped volumes is very important.

Currently, scaling round-robin striped (RR-striped) volumes is usually done in one of the following ways.

—Offline redistribution. To add disks, the application has to be stopped first, then all data chunks[1] on the volume are redistributed across the original and new disks with a reconfiguration tool like raidreconf [Stergaard 2001], and finally the application can be reloaded. Apparently, this solution can improve both access bandwidth and storage capacity, but it has a long downtime that may not be affordable in many systems.

—Online concatenation. With a toolkit like LVM2 [Lewis 2005], an RR-striped volume can be extended by concatenating another set of disks onto the end of the first set. This makes it unnecessary to move data chunks on the volume but has the drawback that only storage capacity is increased and access bandwidth is not improved.

[1]The chunk is the basic unit of data striping in RR-striped volumes, and the chunk size (a.k.a. the "stripe unit size") is a multiple of the basic disk block size.

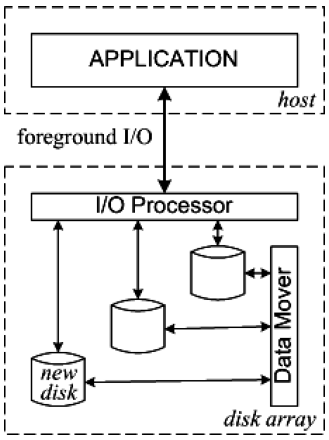

Fig. 1. Senario for online redistribution (a new disk is added).

In short, both offline redistribution and online concatenation have their obvious drawbacks. Therefore, scaling an RR-striped volume necessitates that the data on the volume be redistributed across the original and new disks without interruption to the activity of the applications.

In this article, we investigate the problem of scaling RR-striped volumes online and propose an efficient approach. Round-robin striping mainly includes widely-used RAID-0, RAID-4, and RAID-5. We first concentrate on the situation where some disks are added into a RAID-0 volume, then propose our approach, and evaluate it in detail. Further, we conclude that our approach can be extended for not only disk removal from a RAID-0 volume but also disk addition/removal in a RAID-4 or RAID-5 volume through discussion.

1.2 Challenge and Strategy

Although scaling RR-striped volumes seems to be a good solution, the process of disk addition is a difficult technical challenge for two reasons. First, almost all data chunks have to be moved to preserve the round-robin order when new disks are added. Second, many computer systems used in e-business, scientific computation, or Web environments depend on the uninterrupted availability of data stored in their storage systems.

Figure 1 shows the senario for online redistribution where the data mover performs data redistribution caused by scaling a striped volume, and the I/O processor serves foreground I/O requests. Solving the problem of scaling RR-striped volumes can be accomplished by finding an efficient approach to redistributing the data, so that:

—data redistribution can be completed in a short time,
—small performance overhead is brought to foreground I/Os, and
—data consistency can be guaranteed during the scaling process.

Without the guidance of the characteristics in data redistribution, the traditional approach to scaling RR-striped volumes online makes data redistribution

costly. First, it manages mapping information with the mapping table [Kim et al. 2001] that expands with the volume size; second, it writes mapping metadata onto the disk for each data chunk movement; third, it reads or writes only one data chunk via each I/O.

In order to present a better solution to the problem of scaling striped volumes, we shall analyze the data redistribution process first. Data redistribution requires (1) reads and writes of data chunks and (2) updates of mapping metadata. Although it is impossible to decrease the number of data chunks moved while preserving the round-robin order, [Seo and Zimmermann 2005], is it also impossible to reduce the reads/writes of data chunks and updates of mapping metadata? We focused on reducing the cost of data redistribution by decreasing the number of chunk reads/writes and that of metadata updates without enlarging the negative impact on foreground I/Os.

1.3 Contributions of This Article

This article makes two main contributions in solving the problem of scaling RR-striped volumes.

(1) *Presenting and proving a reordering window characteristic in scaling RR-striped volumes*. We found and proved that during the data redistribution process, there always exists a reordering window where data consistency can be maintained, while changing the order of data movements. Some useful conclusions can be drawn from the concept of a reordering window such as the following.
   a) For almost every data chunk, its mapping information does not have to be updated as soon as it is copied to its new location in order to guarantee data consistency.
   b) If and only if data chunks coexist in the same reordering window is the order of their movements changeable.
   c) A data chunk in a reordering window may have two valid replicas when the chunk has been copied to its new location and has not been written since it was copied.

   With the importance of these conclusions in solving the problem of scaling RR-striped volumes, the reordering window characteristic provides a theoretical basis for solving this problem.

(2) *Proposing an efficient approach to scaling RR-striped volumes*. Taking advantage of the reordering window characteristic, we propose SLAS (an acronym for Sliding window, Lazy updates and movement Scheduling), an efficient approach to scaling RR-striped volumes. The approach includes three major strategies.
   a) SLAS applies a new mapping-management solution based on a sliding window, which not only occupies a very small space but also enables newly added disks to be gradually available to serve I/O requests during the scaling process. The mapping-management solution makes it convenient for SLAS to make use of the reordering window characteristic.

b) SLAS uses lazy updates of mapping metadata to decrease the number of metadata writes required by data redistribution. Supported by the reordering window characteristic, lazy updates of mapping metadata can still keep data consistent.
c) SLAS also changes the order of data chunk movements to read/write multiple data chunks via an I/O. This can reduce the cost of data redistribution because I/O aggregation can decrease the number of disk seeks.

We implemented a detailed simulator that uses disksim as a slave module to simulate disk accesses. Under several real-system workloads like Cello-92, TPC-C, and Cello-96, we evaluated the traditional approach and our SLAS approach. The experimental results demonstrate that, compared with the traditional approach, SLAS shortens redistribution duration by up to 40.79% without enlarging the maximum response time of foreground I/Os. Additionally, the SLAS approach also guarantees data consistency and, therefore, enables striping to survive panics and power failures.

Finally, we prove that there is also the reordering window characteristic in disk addition/removal in a RAID-4 or RAID-5 volume. Therefore, our SLAS approach can be extended for scaling a RAID-3 or RAID-5 volume.

1.4 Roadmap to the Remainder of this Article

The remainder of this article is organized as follows. Section 2 examines the research background and related work. The problem of scaling striped volumes and the traditional approach are described in Section 3. Section 4 presents the reordering window characteristic in the data redistribution process. Section 5 proposes SLAS, an effective solution to the problem of scaling RR-striped volumes. Section 6 evaluates the SLAS approach through detailed simulation experiments. The extendability of our SLAS approach is discussed in Section 7. Finally, Section 8 summarizes our research and discusses future research work.

2. BACKGROUND AND RELATED WORK

2.1 Widely-Used Round-Robin Striping

Many disk arrays and logical volume managers stripe data across their disks in a round-robin fashion. There have been some efforts on how to utilize RR-striped volumes to obtain better I/O performance. Most of them concentrate on choosing a chunk size to provide effective load balancing of small I/Os and parallel transfers for large I/Os [Schindler et al. 2004; Chen and Patterson 1990; Livny et al. 1987]. A typical choice for the chunk size is 32-64KB [Hennessy and Patterson 2003; Wilkes et al. 1996]. In addition, staggered striping [Berson et al. 1994] minimizes the percentage of disk bandwidth that is wasted when the storage server consists of heterogeneous media objects with different bandwidth requirements. Nevertheless, all such efforts do not solve the problem of scaling RR-striped volumes.

2.2 Limitations of Random Striping

Disk addition into RR-striped volumes online remains a difficult technical challenge, thus random striping is now gaining the spotlight in the data placement area. Random allocation [Alemany and Thathachar 1997; Goel et al. 2002] across the disk set appears to have better scalability, but it has some limitations. The simulation report in Alemany and Thathachar [1997] shows that a single copy of the data in random striping may result in some hiccups of the continuous display. Although a randomized hash function called SCADDAR [Goel et al. 2002] is proposed, the function does not preserve the randomness of data layout after several disk additions or deletions [Seo and Zimmermann 2005]. Since random striping is not so satisfactory as expected, round-robin striping is still an important data organization method.

2.3 Disk Addition into RR-Striped Volumes

The mapping-function solution [Teigland and Mauelshagen 2001] does not support online data redistribution caused by scaling RR-striped volumes. The mapping table [Kim et al. 2001] can handle such online redistribution, but its size increases with the volume size. Our mapping management solution, based on a sliding window [Zhang et al. 2005], not only enables newly added disks to be gradually available to serve I/O requests during the scaling process, but also occupies a very small space.

Shahram and Dongho [1996] proposed that a system should employ both eager and lazy reorganizations to redistribute data online. However, this policy requires a large amount of extra disk storage (60% of the total capacity of original disks at most) to maintain multiple copies of migrated chunks. The TH-VSS system [Xiao et al. 2005] employs a mirroring mechanism to ensure that the data on striped volumes is not corrupted during online redistribution. The system is difficult to apply because it requires that all the disks in the original striped volume have a large unused storage space. Unlike these systems, SLAS requires no additional disk space, and data consistency is guaranteed.

The HP AutoRAID [Wilkes et al. 1996] migrates data between RAID-1 and RAID-5, depending on its current access frequency. It allows an online capacity expansion and automatically takes advantage of the additional space by allocating more mirrored storage. But the system cannot add new disks into an existing RR-Striped (RAID-5) volume online.

2.4 Rate Control in Data Movement

Gonzalez and Cortes [2004] proposed an algorithm for increasing the capacity of RAID5, which has an easily controlled overhead. There are still some other efforts focusing on the rate control problem of online data migration. Lu et al. [2002] and Verma et al. [2005] proposed a control-theoretic approach to control the rate of data migration. Dasgupta et al. [2005] formulated the data migration problem as a reward maximization problem and presented an approach to migrating data. The goal of rate control is to gain a reasonable trade-off between

Table I. Symbolic Notations

Symbol	Meaning
A_n^m	a request to add m disks into a striped volume made up of n disks
R_n^m	a request to remove m disks from a striped volume made up of n disks
$S(n,m,x)$	the disk-scaling state of A_n^m or R_n^m at Chunk x
$ROW(n,m,x)$	the size of the reordering window of A_n^m or R_n^m at Chunk x
$G(n,m,x)$	the operation granularity of A_n^m or R_n^m at Chunk x
$OD(x)$	the ordinal number of the disk which Chunk x lies on before scaling
$OE(x)$	the ordinal number of the physical chunk which Chunk x lies on before scaling
$ND(x)$	the ordinal number of the disk which Chunk x lies on after scaling
$NE(x)$	the ordinal number of the physical chunk which Chunk x lies on after scaling

the migration objectives and the I/O performance requirements. However, the purpose of our SLAS approach is to reduce the cost of data redistribution caused by scaling RR-striped volumes.

3. PROBLEM DESCRIPTION AND TRADITIONAL APPROACH

3.1 Problem Description

Given an RR-striped volume made up of n disks, a request to add m disks and transform the $n + m$ disks into a new RR-striped volume, during the time when RAID storage is still functional, is termed a *disk-scaling request* and is represented as A_n^m. A disk-scaling request requires almost all data chunks to be redistributed online to preserve the round-robin order.

The problem of scaling RR-striped volumes can be described as follows. Given a disk-scaling request A_n^m, the goal is to find an effective approach to data redistribution so that the objectives of Objective 1 and Objective 2 are met under the restriction of Restriction 1.

Objective 1. The data redistribution can be completed in a short time.

Objective 2. The data redistribution can make a very small negative impact on the performance of foreground I/Os.

Restriction 1. Data consistency is guaranteed during redistribution so that striping can survive panics and power failures.

Table I summarizes the symbolic notations used in this article.

3.2 Traditional Approach

The traditional approach to the problem of scaling RR-striped volumes uses the mapping-table solution to manage mapping information [Kim et al. 2001]. After getting the access permission of the mapping table, the data mover copies a data chunk to its new location and writes the corresponding metadata onto the disk repeatedly. Because the I/O processor reads the mapping table to locate underlying physical storage and the data mover writes the mapping table to reflect data movements, reader-writer locks have to be provided for the mapping table to guarantee data consistency. As a trade-off between I/O parallelism and locking overhead, the traditional approach divides the whole mapping table into several regions and provides a reader-writer lock for each region.

The general steps the traditional approach follows are as follows.

Data mover
1. repeat until all the data is redistributed
    1.1) acquire writer lock for the related region
    1.2) for 1 to n do
        1.2.1) read another next chunk from its original location
        1.2.2) write this chunk to its new location
        1.2.3) update mapping metadata on the disk for the movement
    1.3) release the writer lock acquired for these movements

I/O Processor
Suppose that data b is accessed
1. acquire all reader locks for the regions that data b overlaps
2. according to mapping metadata, access each chunk in its original or new location for data b
3. release all reader locks acquired for this I/O

Without the guidance of the reordering window characteristic, the data mover in the traditional approach moves only one data chunk via a pair of read and write accesses because changing the movement order of any other fixed number of data chunks will cause overwriting of valid chunks. This makes data redistribution costly. First, it writes mapping metadata onto the disk once for each chunk movement. Second, it reads or writes only one data chunk via an I/O.

4. A REORDERING WINDOW DURING DATA REDISTRIBUTION

Our initial thought is to solve the problem of scaling striped volumes by scheduling the data redistribution process. If data chunk movements have to be completed in a strict one-by-one sequence, the data redistribution process will be unschedulable. Fortunately, we find that during the data redistribution process, there is always a reordering window where no valid data chunk will be overwritten while changing the order of data movements. Because any chunk-copying in a reordering window does not overwrite any valid data, even if the power fails before the mapping metadata is updated, only some chunk reads/writes are wasted, and data consistency is not destroyed since the chunk in its original location is valid. Therefore, a reordering window is a window where data consistency can be maintained while changing the order of chunk movements.

Definition 1. Given a disk-scaling request A_n^m and a chunk x, the fact that Chunk x' (for $\forall x'\{0 \leq x\prime < x\}$) has been moved and Chunk x'' (for $\forall x''\{x'' \geqslant x\}$) has not been moved is called a disk-scaling state and is represented as $S(n, m, x)$.

A transition from one disk-scaling state to another is called a disk-scaling operation. It transforms from one disk-scaling state $S(n,m,x)$ to another state $S(n,m,x')$, while moving $(x' - x)$ data chunks to new locations. Here, the positive integer $(x' - x)$ is called the operation granularity. The unit of operation granularity is a data chunk.

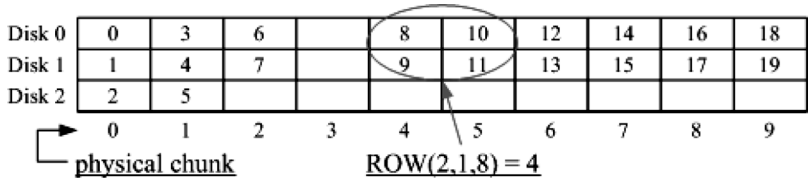

Fig. 2. Illustration of S(2,1,8). (Disk 2 is newly added.)

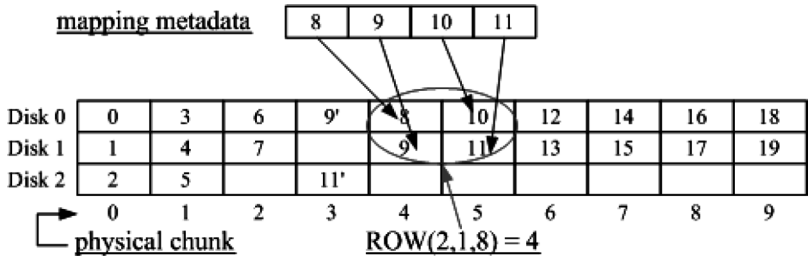

Fig. 3. Guarantee of data consistency. (Chunks 9 and 11 have been copied to their new locations.)

Definition 2. Given a disk-scaling state $S(n, m, x)$, of a disk-scaling request A_n^m, the window of data chunks, whose size equals $MAX\{\delta \mid$, where δ is a nonnegative integer; for $\forall x' \forall x''\{x', x'' \in [x, x+\delta)$ and $x' \overset{\delta}{<} x''\}$, Chunks x' and x'' cannot be overwritten by each other no matter in what order they are moved.$\}$, is called the reordering window of A_n^m at Chunk x. The size of the reordering window is represented as $ROW(n, m, x)$.

Example 1. To clarify the concept of a reordering window, we take the disk-scaling request A_2^1 as an example. Figure 2 illustrates the disk-scaling state $S(2,1,8)$. When Chunks 8, 9, 10, and 11 are moved in 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, ROW(2,1,8)=4.

Further, suppose that Chunks 9 and 11 have been copied to their new locations (see Figure 3), the power fails, or the system crashes before the mapping metadata is updated. The original replicas of Chunks 9 and 11 will be used after the system reboots. Only if Chunks 9 and 11 have not been written since they were copied, will data consistency still be guaranteed.

In the same way, we can get the results as follows.

ROW(2,1,0)=0;	ROW(2,1,1)=0;	ROW(2,1,2)=1;	ROW(2,1,3)=1;
ROW(2,1,4)=2;	ROW(2,1,5)=2;	ROW(2,1,6)=3;	ROW(2,1,7)=3;
ROW(2,1,8)=4;	ROW(2,1,9)=4;	ROW(2,1,10)=5;	ROW(2,1,11)=5;
ROW(2,1,12)=6;	ROW(2,1,13)=6;	ROW(2,1,14)=7;	...

Inducing the above results, we get ROW(2,1,x) = $\lfloor x / 2 \rfloor$. Generally speaking, $ROW(n,m,x)$ can be gained according to Lemma 1.

LEMMA 1. $ROW\ (n,m,x) = m \times \lfloor x/n \rfloor$.

PROOF. As for $S(n, m, x)$, let $OD(x) = d$, $OE(x) = e$, where both d and e start at 0. We have $e = \lfloor x/n \rfloor$, $d = x\%n$. Let us assume that the logical chunk x' will be placed on chunk e of disk d in the new configuration, then $ND(x') = d$, $NE(x') = e$, and $x' = e \times (n + m) + d = \lfloor x/n \rfloor \times (n + m) + (x\%n)$. Further, we have $\delta = x' - x = \lfloor x/n \rfloor \times (n + m) + x\%n - x = m \times \lfloor x/n \rfloor$. In the following, we prove that δ is just the size of the reordering window of A_n^m at Chunk x by contradiction.

First, we assume that, $\exists\, x_a\ \exists\, x_b\{x_a, x_b \in [x, x + \delta)$ and $x_a < x_b\}$; when data chunks x_a, x_b are moved in some order, one will be overwritten by the other.}. It is evident that no chunk will be overwritten if Chunk x_a is moved first. So, Chunk x_b is moved first and overwrites Chunk x_a. Therefore, $x_b = \lfloor x_a/n \rfloor \times (n + m) + (x_a\%n)$. Since $x_a \geq x$, we have $x_b \geq x' = x + \delta$. This result contradicts our initial assumption that $x_b \in [x, x + \delta)$. Therefore, for $\forall x_a \forall x_b\{x_a, x_b \in [x, x + \delta)$ and $x_a < x_b\}$, data chunks x_a and x_b cannot be overwritten by each other no matter in what order they are moved.

Additionally, for arbitrary nonnegative integer $\delta'\{\delta' > \delta\}$, we have $x, x' \in [x, x + \delta')$. If Chunk x' is moved before Chunk x, it will overwrite Chunk x.

According to Definition 2, δ is the size of the reordering window of A_n^m at Chunk x. That is, ROW(n,m,x) = δ = m $\times$ $\lfloor$x / n$\rfloor$. □

According to Lemma 1, ROW(2,1,x) = $\lfloor$x / 2$\rfloor$. This result accords with Example 1.

Deduction 1. Given n and m, $ROW(n,m,x)$ increases with x in a step manner.

Deduction 2. During data redistribution caused by scaling RR-striped volumes, all data chunks which do not coexist in the same reordering window cannot be moved in arbitrary order because this will cause valid data to be overwritten.

The reordering window characteristic provides a theoretical basis for solving the problem of scaling RR-striped volumes. For example, the following conclusions can be drawn from the concept of a reordering window:

(1) For almost every data chunk, its mapping information does not have to be updated immediately after it is copied to its new location, while meeting the restriction of data consistency.
(2) If and only if data chunks coexist in the same reordering window is their movement order changeable.
(3) 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.
(4) Multiple data chunks in a reordering window can be moved concurrently for scaling the striped volumes.

These conclusions are of great importance to solving the problem of scaling striped volumes. For instance, the first three conclusions are used in our SLAS approach.

5. OUR SOLUTION TO SCALING STRIPED VOLUMES

Using the reordering window characteristic, we propose an efficient approach to scaling striped volumes called SLAS. First, SLAS uses a new mapping-management solution based on a sliding window, which not only occupies a very small space, but also enables newly added disks to be gradually available to serve I/O requests during the scaling process. Second, SLAS uses lazy updates of mapping metadata to decrease the number of metadata writes required by data redistribution. Finally, SLAS also changes the order of data chunk movements to read/write multiple data chunks via an I/O.

5.1 Mapping Management

Mapping metadata is used to map a logical address of a striped volume to a physical address of an underlying physical disk. There have been some traditional solutions to mapping management. However, they are inadequate for scaling striped volumes online in large-scale storage environments. The discovery of the reordering window helps improve mapping-management for striped volumes. Our SLAS approach applies a new mapping-management solution based on a sliding window, which requires a very small space and makes it possible to take full advantage of the reordering window characteristic.

5.1.1 *Limitations of Traditional Solutions to Mapping Management.* There are already two traditional solutions to mapping management: mapping function [Teigland and Mauelshagen 2001] and mapping table [Kim et al. 2001; Lim et al. 2003]. Mapping function takes a logical address referenced by an I/O request as the input parameter and the corresponding physical address as the output parameter. The mapping-function solution only stores the mapping function and therefore occupies a very small space. With this technique, however, foreground I/O operations are limited at data redistribution. The I/O operation which occurs during the data redistribution cannot find the correct location of relevant data because the data can exist in either its original or new location.

A mapping table for a striped volume is a one-dimensional array of the form MT[t]. Each entry in the mapping table contains a physical disk identifier and a chunk offset in the physical volume. The mapping-table solution can handle data redistribution and foreground I/O operations at the same time since it can keep track of the movement of data. However, unlike the mapping function, which only stores its own function, the mapping table occupies a large space that expands with the volume size. Moreover, the smaller the chunk is, the larger the mapping table is. A typical volume manager is implemented in the kernel space of a host. Maintaining large amounts of mapping information in the host memory may reduce the amount of main memory available for I/O cache and further impair the I/O performance of the host.

5.1.2 *Our Mapping Management Solution Based on a Sliding Window.* According to Deduction 2, data chunk movements caused by scaling striped volumes are relatively sequential. This makes it possible to save the space storing mapping information. We propose a new mapping-management solution based on a sliding window. The key idea behind the solution is to introduce the concept of a sliding window into the mapping function. Our solution is equal to

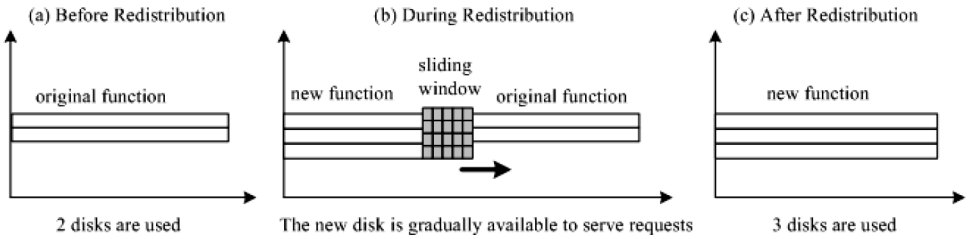

Fig. 4. Mapping management based on a sliding window for the data redistribution caused by A_2^1.

the mapping-function solution when data redistribution is not needed, while a sliding window is introduced when the data needs to be redistributed. The sliding window is similar to a small mapping table, and it describes the mapping information of a continuous segment of the striped volume.

Figure 4 illustrates the mapping-management solution based on a sliding window for the data redistribution caused by A_2^1. Before the data redistribution, the original mapping function is used, and 2 disks are used to serve requests. During the data redistribution, only data within the range of the sliding window are redistributed. The foreground I/O requests, sent to the logical address in front of the sliding window, are mapped through the original function; those sent to the address behind the sliding window are mapped through the new function; and those sent to the address in the range of the sliding window are mapped through the sliding window. After all of the data in the sliding window are moved, the window slides ahead by one window size. Thus, the newly added disk is gradually available to serve foreground I/O requests. The data redistribution of the whole volume is completed when the sliding window reaches the end of the original striped volume. From then on, the address mapping of the whole volume is performed through the new mapping function and 3 disks are used to serve requests.

It should be noted that a sliding window is different from a reordering window. The former is a mapping-management solution for striped volumes, while the latter is a characteristic in the process of scaling RR-striped volumes. It can be shown from Section 5.2 that the mapping-management solution based on a sliding window is convenient for making use of the reordering window characteristic to solve the problem of scaling striped volumes. To meet the restriction of data consistency, the size of a sliding window (i.e. operation granularity) should not be greater than that of the corresponding reordering window.

If the maximum of the operation granularity is set to *Max_Granularity*, we can dynamically get $G(n,m,x)$, the operation granularity of a disk-scaling operation whose starting disk-scaling state is $S(n,m,x)$, according to the following equation:

$$G(n,m,x) = \min(ROW(n,m,x), Max\_Granularity). \tag{1}$$

Because $ROW(n,m,x)$ increases with x in a step manner (see Deduction 1), $G(n,m,x)$ will equal *Max_Granularity* eventually. Therefore, setting *Max_Granularity* is also called choosing an operation granularity.

The mapping-management solution based on a sliding window enables the newly added disks to be gradually available to serve requests during the

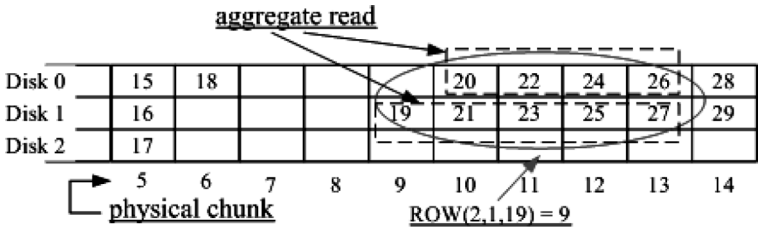

Fig. 5. Aggregate reads for $S(2, 1, 19)$.

disk-scaling process. Additionally, it brings higher performance and better scalability because a sliding window occupies a very small space and its size is independent of the size of the whole striped volume.

5.2 Our SLAS Approach

By using the reordering window characteristic, we can work out some approaches to scaling striped volumes. In this section, we first give the NPP approach in which a disk-scaling operation cannot be preempted by a foreground I/O operation, and we analyze its limitations. Further, we propose the SLAS approach in which a disk-scaling operation is preemptive. It should be noted that a disk-scaling operation moves all the data chunks in a sliding window. There are two main ideas in SLAS to improve the redistribution efficiency. The first idea is to decrease the number of metadata writes. The second idea is to aggregate multiple reads/writes of data chunks into an I/O.

5.2.1 *The NPP Approach Using the Nonpreemptive Policy.* Our initial idea to make use of the reordering window characteristic to scale a striped volume is the NPP (an acronym for NonPreemptive Policy) approach, in which disk-scaling operations are nonpreemptive, that is, a disk-scaling operation moves all the data chunks in a sliding window uninterruptedly, and then updates mapping metadata onto the disk. During the whole operation, all foreground I/O requests whose target addresses are within the sliding window will queue for processing.

A disk-scaling operation will not be interrupted, so those data chunks in the current sliding window can be moved in arbitrary order. NPP changes the movement order of data chunks in a sliding window so as to aggregate reads/writes of multiple data chunks. Take the disk-scaling state $S(2,1,19)$ as an example to make the aggregation technique clear; we get $\mathrm{ROW}(2,1,19) = 9$. As shown in Figure 5, NPP issues an I/O request to read Chunks 20, 22, 24, and 26, another I/O request to read Chunks 19, 21, 23, 25, and 27. Thus the NPP approach requires two I/Os instead of nine to read these chunks.

When all these chunks have been read into memory, NPP issues an I/O request to write Chunks 21, 24, and 27, a second one to write Chunks 19, 22, and 25, a third one for Chunks 20, 23, and 26 (see Figure 6). In this way, only three instead of nine write requests are issued.

In the previous case, nine chunks will reside in memory at some time, so the memory space to hold nine chunks will have to be reserved. The set of

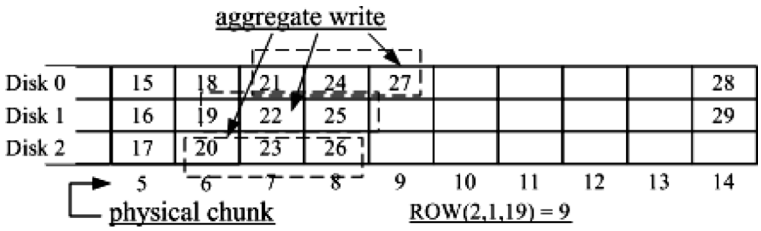

Fig. 6. Aggregate writes for $S(2, 1, 19)$.

the data chunks that the reserved memory can hold is called a subwindow. In fact, the size of reserved memory is tunable. Suppose that the size of the subwindow is six, the first six chunks (i.e., Chunks 19 to 24) will first be read into memory. Then the first aggregate read request will read Chunks 20, 22, and 24; the first write request will write Chunks 21 and 24. To move all these nine chunks, four reads and six writes will be required. Given the number of the disks constructing a striped volume, the ordinal number of a start logical chunk, and the size of a subwindow, we can calculate the offset and size of an aggregate read/write for each disk.

Because the NPP approach only needs to judge whether the target address of a foreground I/O request is in the sliding window, it maintains a window offset and a window size instead of the whole sliding window. The offset and the size are protected by a reader-writer lock because the I/O processor needs to read them while the data mover needs to write them.

The general steps the NPP approach follows are as follows.

Data mover
1. initialize the first sliding window
2. repeat until all the data is redistributed
    2.1) acquire writer lock for the sliding window
    2.2) for each sub-window of data chunks in the current sliding window
        2.2.1) for each disk in the original configuration
            2.2.1.1) read data chunks in the sub-window aggregately
        2.2.2) for each disk in the new configuration
            2.2.2.1) write data chunks in the sub-window aggregately
        2.3) update mapping metadata for the movement of the sliding window
        2.4) release the writer lock acquired for this sliding window
        2.5) initialize the next sliding window

I/O Processor
Suppose that data b is accessed
1. if data b is before the sliding window
    1.1) access data b in its original location
2. if data b is after the sliding window
    2.1) access data b in its new location
3. if data b overlaps the sliding window
    3.1) acquire reader lock for the sliding window
    3.2) access the portion of data b before the sliding window in its original location, and access the other portion of data b in its new location
    3.3) release reader lock acquired for this I/O

As for the NPP approach, it is a dilemma to choose an operation granularity to meet both the redistribution duration objective and the I/O performance objective. If the operation granularity is set to 1, it will result in a larger cost of data redistribution for two reasons. First, one write of mapping metadata onto the disk is required for each data chunk movement. Second, there is no chance to aggregate reads/writes of data chunks.

Increasing the operation granularity allows multiple chunk movements to be processed with only one write of mapping metadata onto the disk and multiple chunks to be read/written in an aggregate I/O. This can reduce the cost of data redistribution. In other words, by increasing the operation granularity, the redistribution duration objective can be better met. In addition, when increasing the operation granularity, the restriction of data consistency remains guaranteed. Even if the system crashes in a disk-scaling operation, data copying in a disk-scaling operation cannot overwrite any valid data because the operation granularity is not greater than the size of the reordering window; data in the sliding window cannot be updated by any foreground I/O due to the nonpreemptive policy, therefore, data consistency can be guaranteed. Unfortunately, the larger the operation granularity is, the more difficult it is to meet the I/O performance objective. Increasing the operation granularity will cause more I/O requests to queue up at the sliding window, which will build up the foreground I/O latency.

In a word, the NPP approach is not a satisfactory solution to the problem of scaling striped volumes.

5.2.2 *The SLAS Approach Using the Preemptive Policy.* Due to the nonpreemptive policy, the NPP approach has an insurmountable limitation. Our improvement on the NPP approach is the SLAS approach using the preemptive policy. Similar to NPP, SLAS also changes the movement order of data chunks in a sliding window in order to aggregate reads/writes of multiple data chunks (see Section 5.2.1). Unlike NPP, SLAS can serve foreground I/O requests between aggregate chunk reads/writes in a disk-scaling operation. Because the foreground I/Os, coming between SLAS may be write requests whose target data is in the current sliding window, SLAS cannot simply write mapping metadata after all chunks in the sliding window are moved. SLAS uses lazy updates of mapping metadata to decrease the number of metadata writes caused by data redistribution. While NPP only stores the offset and the size of the sliding window, SLAS stores all of the contents of the sliding window besides the offset and the size.

The technique of lazy updates of mapping metadata is that during a disk-scaling operation, the sliding window will not be updated onto the disk until it has to be updated to meet the restriction of data consistency. Data chunks are copied to new locations in some order, but updates of mapping metadata are done only under one of two circumstances: (1) when all the data chunks in the sliding window have been copied to new locations, the window offset and the size on the disk are updated; (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.

As a by-product of the preemptive policy and read/write aggregation, SLAS has an opportunity to use two tricks to enhance the performance of foreground read I/Os. The first trick is that if a foreground I/O reads a data chunk that has been read in memory for read/write aggregation, the data can be read from memory immediately. The second trick is described as follows. It can be acquired from the reordering window characteristic that chunk x 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 (i.e., $ND(x) \neq OD(x)$), read requests to Chunk x are alternated between the two disks. Although data prefetching and alternating reads from multiple replicas are both widely-used solutions, it is the reordering window characteristic that enables SLAS to reap the benefits of data prefetching and read alteration without any additional labor to prepare in-memory data or multiple replicas.

Another issue to be considered is the data structures that need to be maintained on the disk. The offset and the size of the sliding window needs to be stored, and three more bits also need to be stored for each chunk in the sliding window to denote, respectively, whether the corresponding chunk has been read in memory, copied to its new location or whether its new replica has been updated. Consequently, 1KB of disk space can store the mapping metadata for 2,698 (1KB/ 3bits $= (1024*8)/3 = 2698$) chunks.

The general steps the SLAS approach follows are as follows.

Data mover

1. initialize the first sliding window
2. repeat until all the data is redistributed
    - 2.1) for each sub-window of data chunks in current sliding window
        - 2.1.1) for each disk in the original configuration
            - 2.1.1.1) acquire writer locks for the chunks in the sub-window that locate on the disk
            - 2.1.1.2) read data chunks in the sub-window aggregately
            - 2.1.1.3) write a memory record to indicate these data chunks has been read in memory
            - 2.1.1.4) release the writer locks acquired for this aggregate read
        - 2.1.2) for each disk in the new configuration
            - 2.1.2.1) acquire writer locks for the chunks in the sub-window that locate on the disk
            - 2.1.2.2) write data chunks in the sub-window aggregately
            - 2.1.2.3) write a record to indicate these data chunks has been copied to new locations
            - 2.1.2.4) release the writer locks acquired for this aggregate write
        - 2.1.3) clean up the memory records indicating these data chunks are in memory
    - 2.2) update mapping metadata for the current window
    - 2.3) initialize the next sliding window

I/O Processor

```
Suppose that data b is accessed
1. if data b is before the sliding window
    1.1) access data b in its original location
2. if data b is after the sliding window
    2.1) access data b in its new location
3. if data b overlaps the sliding window
    3.1) acquire reader locks for the data chunks of data b in the
         sliding window
    3.2) if this is a read request
        3.2.1) read the portion of data b before the sliding window from
               its original location
        3.2.2) read the portion of data b after the sliding window from
               its new location
        3.2.3) for each chunk x of data b in the sliding window
            3.2.3.1) if chunk x is in memory
                3.2.3.1.1) read chunk x from memory
            3.2.3.2) if chunk x is not copied yet
                3.2.3.2.1) read chunk x from original location
            3.2.3.3) if chunk x is copied to new location and is written
                     since it is copied
                3.2.3.3.1) read chunk x from new location
            3.2.3.4) if chunk x is copied to new location and is not
                     written since it is copied
                3.2.3.4.1) read chunk x from the lightly-loaded one of
                           the two disks
    3.3) if this is a write request
        3.3.1) write the portion of data b before the sliding window to its
               original location
        3.3.2) write the portion of data b after the sliding window to its
               new location
        3.3.3) if any chunk of data b in the sliding window is copied and
               is not written since it is copied
            3.3.3.1) update mapping metadata on the disk to indicate
                     the movement
        3.3.4) for each chunk x of data b in the sliding window
            3.3.4.1) if chunk x is in memory
                3.3.4.1.1) write chunk x to the replica on memory
            3.3.4.2) if chunk x is not copied yet
                3.3.4.2.1) write chunk x to original location
            3.3.4.3) if chunk x is copied to new location
                3.3.4.3.1) write chunk x to new location
    3.4) release reader locks acquired for this I/O
```

The preemptive policy not only eliminates the problem of many I/O requests queuing up at the sliding window which is intrinsic to the nonpreemptive policy, but also makes it possible to reduce the cost of data redistribution and the response time of foreground I/O requests through some optimizations.

Ordered operations of copying a data chunk and updating the sliding window can ensure data consistency of striping [Zhang et al. 2005]. But ordered operations cause each chunk movement to require one write of the sliding window, which results in a large cost of data redistribution. Luckily, the reordering window characteristic enables data consistency to also be guaranteed through lazy updates. One write of the sliding window can store multiple map

changes of data chunks, so lazy updates can decrease the number of metadata writes significantly. In the best case, a disk-scaling operation only requires one write of the offset and the size without any write of the sliding window. Therefore, lazy updates reduce the cost of data redistribution with guaranteed consistency.

The reordering window characteristic makes the movement order of data chunks in a sliding window changeable. Read/write aggregation based on movement scheduling enables SLAS to have larger redistribution throughput. A typical choice for the chunk size is 32KB (EMC's Symmetrix 8000 [Hennessy and Patterson 2003]) or 64KB (HP AutoRAID [Wilkes et al. 1996]). So increasing the size of reads/writes will enhance the I/O throughput.

Using in-memory chunks and alternating read requests help improve the performance of read requests to data chunks in a sliding window. Generally speaking, a large fraction of file accesses are read-only. For instance, in his study of Windows NT 4.0 file system behavior, Vogels [1999] found that 79% of accesses to files were read only. Especially for some specific applications like media servers, nearly all the I/O accesses are read only. Thus, these two tricks can to some extent compensate for the performance penalty that data redistribution brings to foreground I/Os.

To sum up, the SLAS approach, which takes advantage of the reordering window characteristic, is a satisfactory solution to the problem of scaling RR-striped volumes. First, it meets the redistribution duration objective through lazy updates of mapping metadata and aggregate reads/writes of data chunks. Second, it meets the I/O performance objective through the preemptive policy and by two tricks for improving read performance. Last but not least, it also meets the restriction of data consistency.

6. EXPERIMENTAL EVALUATION

6.1 Evaluation Methodology

To evaluate the benefits of exploiting the SLAS approach in scaling RR-striped volumes, we used detailed simulations with several disk traces collected in real systems. Our simulators for different approaches were implemented in SimPy [Vignaux and Muller 2005] and used Disksim [Bucy and Ganger 2003] as a slave module to simulate disk accesses. We did not use the RAID simulator implemented in disksim because it does not support online data redistribution. Each simulator is logically divided into three different parts: a workload generator, an I/O processor, and a data mover. According to trace files, the workload generator initiates an I/O request at the appropriate time so that a particular workload is induced on the striped volume. The I/O processor, according to the mapping metadata, forwards incoming foreground I/O requests to the corresponding disks simulated by disksim. The data mover reorganizes the data on the striped volume.

The simulated disk specification is that of the 7200RPM IBM Ultrastar 18ES which is included in the disksim 3.0 code. Our experiments used the following three real-system traces:

(i) *Cello-92* was collected at Hewlett-Packard Laboratories in 1992 [Ruemmler and Wilkes 1993]. It captured all low-level disk I/O performed on Cello, which is a timesharing system used by a group of researchers at HP Labs to do simulations, compilation, editing, and email. The trace includes the accesses to 8 disks.

(ii) *TPC-C* contains the tracing of HP's Client/Server TPC-C application running at approximately 1150tpmC on a 100 Warehouse database. The KI trace points enqueue and queuedone were traced for approximately 4.2 M I/Os. The system was doing approximately 700 I/Os per second during steady state.

(iii) *Cello-96* is similar to Cello-92. The only difference is that this trace was collected in 1996 and contains more modern workloads. It includes the accesses to 20 disks from multiple users and miscellaneous applications.

In all experiments, the sliding window sizes for the NPP and SLAS approaches were set to 1,024. To provide a fair comparison, we also divided the whole mapping table into 1,024 regions and used 1,024 reader-writer locks in the traditional approach. Each simulation experiment lasted from the beginning to the end of data redistribution. We focused on comparing redistribution durations and maximum response times of foreground I/Os when different approaches were used. To achieve this goal, we collected the elapsed time when every one twentieth of all the data chunks were moved. In addition, we collected the response time of all foreground I/Os, divided the response time sequence evenly into 2,000 sections, and got a local maximum response time (MRT) from each section. The 2,000 local MRTs make up a local MRT series.

6.2 Advantages of the SLAS Approach

The purpose of our first experiment is to quantitatively characterize the advantages of the SLAS approach through a comparison among the traditional, NPP, and SLAS approaches. We simulated the data redistribution caused by a disk-scaling request A_8^1 where each disk had a capacity of 3.2GB. Each approach was performed with the 32KB chunk size under a Cello-92 workload. The reserved 1.28MB memory for chunk read/write aggregation could hold 40 chunks.

The redistribution using each approach moved $8 \times 100000 - 8 = 799{,}992$ chunks. Figure 7 shows a plot of the elapsed times versus the numbers of moved chunks using the three approaches. On each sampling point, the elapsed times using the NPP and SLAS approaches are nearly identical, and they are obviously shorter than that using the traditional approach. When all the data is redistributed, the durations using the traditional, NPP, and SLAS approaches are 12,953.49, 10,471.39 and 10,471.57 seconds, respectively. That is, SLAS has a 19.16% shorter redistribution duration than the traditional approach.

One of the main factors in reducing the redistribution duration using NPP and SLAS is the decline of the metadata update times. We collected the metadata update times using the three different approaches which are shown in Figure 8. The update times using NPP and SLAS were reduced to 0.1045% and 0.1067% of that using the traditional approach. We believe that the extent

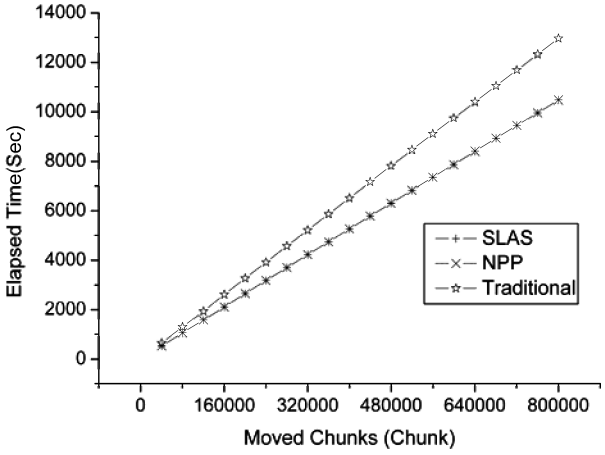

Fig. 7. Elapsed times using the traditional, NPP, and SLAS approaches.

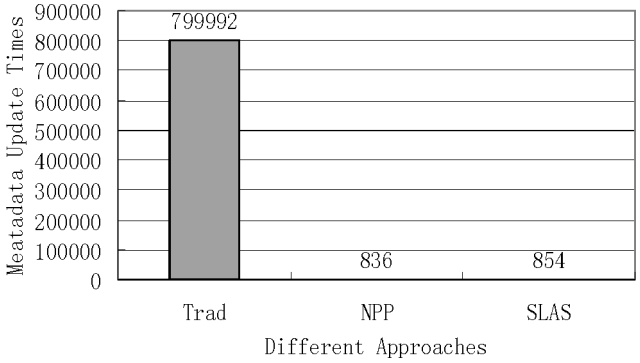

Fig. 8. Metadata update times using the traditional, NPP, and SLAS approaches.

of this decline has tight relations with the size of the sliding window (note: $0.1045\% \approx {}^{1}/_{1024}$ and $0.1067\% \approx {}^{1}/_{1024}$).

Another main factor in reducing the redistribution duration using NPP and SLAS is the decline of the data chunk reads/writes for data redistribution. We also collected the chunk reads/writes using the three different approaches which are shown in Figure 9. The number of chunk reads using the traditional approach is 4.92 times that using NPP and SLAS, and the number of chunk writes using the traditional approach is 4.37 times that using NPP and SLAS. In fact, each aggregate I/O reads $40 \div 8 = 5$ chunks or writes $40 \div 9 \approx 4.44$ chunks on average using the NPP and SLAS approaches.

We compare the three approaches in meeting the redistribution duration objective and analyze the causes for the similar difference in performance improvement for NPP and SLAS compared to the traditional approach. To determine which one of the three approaches meets the I/O performance objective best, Figure 10 plots 2,000 local MRTs versus the completion times of the corresponding I/Os using the three approaches.

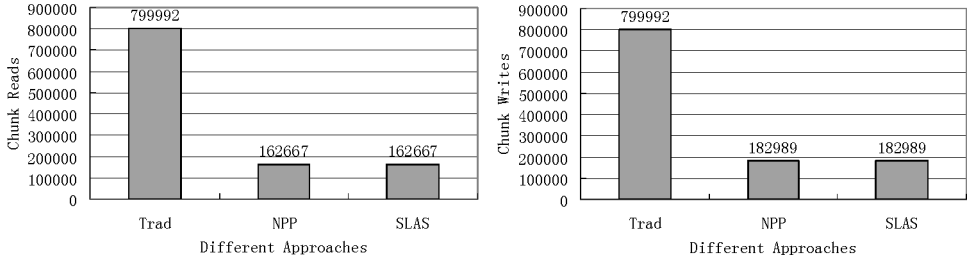

Fig. 9. Read/write numbers for data redistribution using the traditional, NPP, and SLAS approaches.

Although NPP has a shorter redistribution duration than the traditional approach, its local MRTs are obviously longer than those of the traditional approach. The global MRT using NPP even reaches 14.06 seconds. The NPP approach will result in a large number of I/O failures due to response timeout. Of course, the local MRTs will be lower with a decrease in the operation granularity. But the decrease in the operation granularity will necessarily enlarge the redistribution duration. Therefore, NPP is an unsatisfactory solution because it is a dilemma to choose a satisfactory operation granularity for it.

Unlike NPP, even when the operation granularity is as large as 1,024, SLAS has small local MRTs which are similar to those using the traditional approach. Therefore, with both the redistribution duration objective and the I/O performance objective better met, the SLAS approach is a better solution to the problem of scaling RR-striped volumes.

6.3 Benefit of Each Individual Contribution in the SLAS Approach

In Section 6.2, we evaluated the SLAS approach by a comparison with the traditional and NPP approaches and demonstrated the effects of metadata update reduction and chunk read/write aggregation. To make it clearer where the benefits of SLAS come from, we also evaluate each individual contribution in the SLAS approach. Our methodology is to get rid of chunk read/write aggregation from SLAS, leaving only lazy updates of mapping metadata. In each disk-scaling operation, data chunks are read from their original location and written to their new location one-by-one. The setup in the trace-driven simulation is the same as that in Section 6.2.

Figure 11 shows a plot of 2,000 local MRTs versus the completion time of the corresponding I/Os. The local MRTs using SLAS without read/write aggregation are similar to those using the traditional approach (see Figure 10).

Figure 12 shows a plot of the elapsed time versus the number of moved chunks using the three approaches. On each sampling point, the elapsed times using SLAS without read/write aggregation are obviously between those using the traditional and SLAS approaches. Finally, when all the data is redistributed, the durations using the traditional, SLAS without aggregation, and SLAS approaches are 12,953.49, 11,807.86 and 10,471.57 seconds, respectively. That is, SLAS shows an improvement of 19.16% in redistribution duration; lazy updates of mapping metadata contributes 8.84% of the

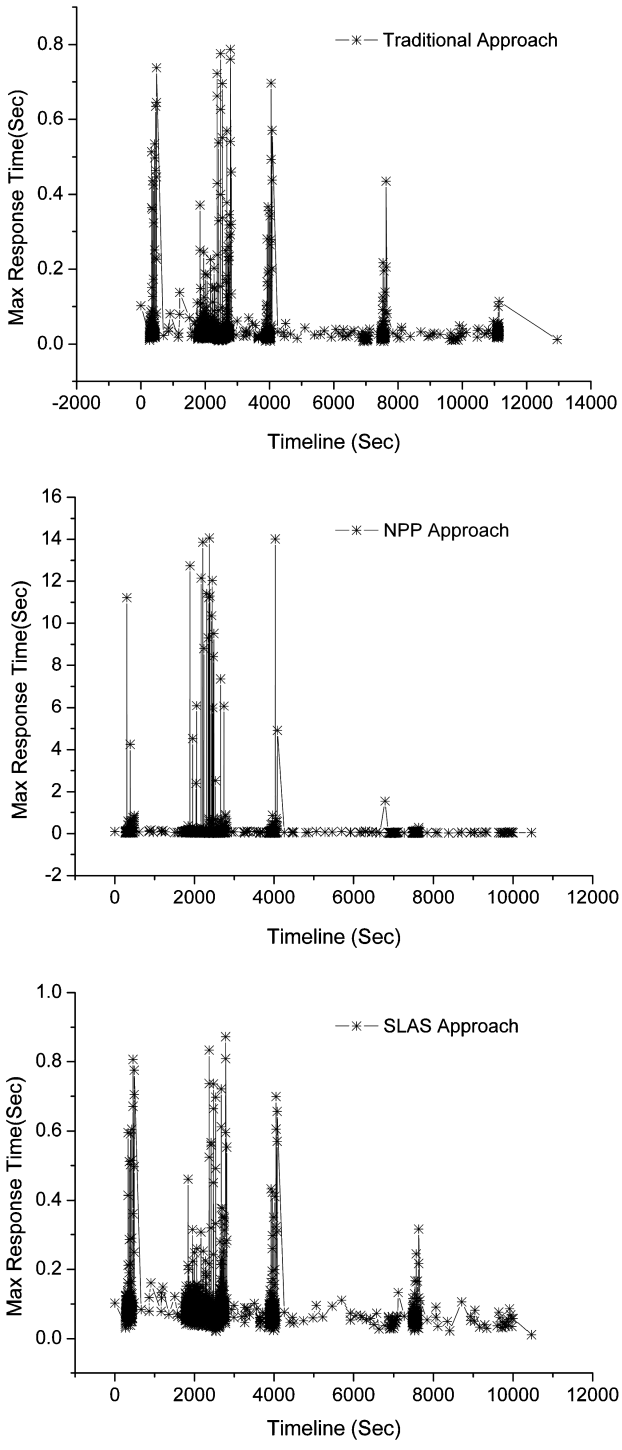

Fig. 10. Local Max-Response-Time series using the traditional, NPP, and SLAS approaches.

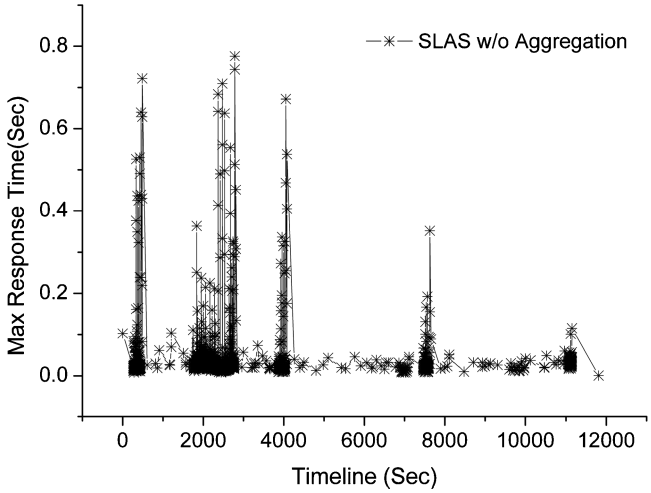

Fig. 11. Local Max-Response-Time series using SLAS without read/write aggregation.

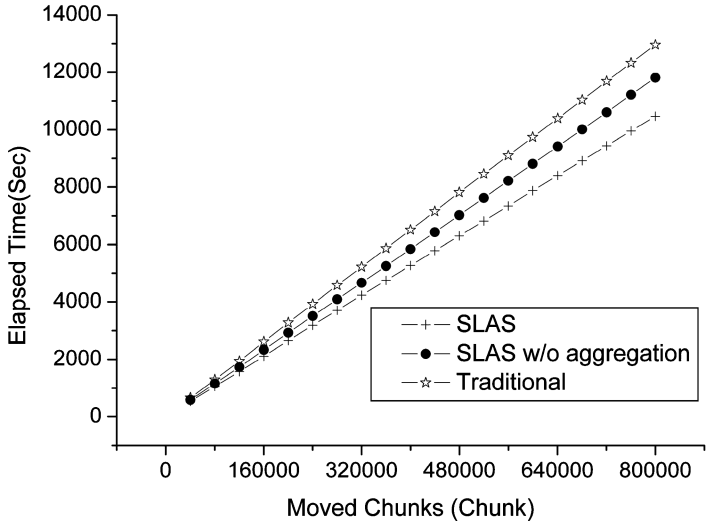

Fig. 12. Elapsed time using the traditional, SLAS w/o aggregation and SLAS approaches.

improvement, and the other 10.32% improvement is contributed by chunk read/write aggregation.

6.4 Impact of the Volume Size

We studied the effects of the volume size by varying the volume size. We measured the performance of the traditional and SLAS approaches to perform disk-scaling requests A_4^1 and A_2^1.

As far as A_4^1 is concerned, Figure 13 plots the elapsed time versus the number of data chunks already moved using the two approaches as the data redistribution proceeded, and Figure 14 shows a plot of 2,000 local MRTs versus the

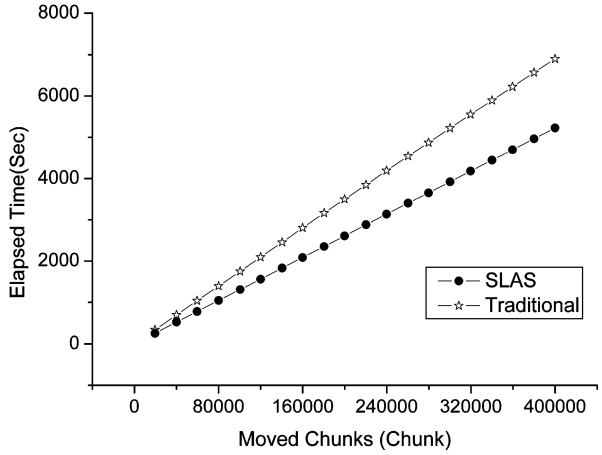

Fig. 13. Elapsed time using the traditional and SLAS approaches for A_4^1.

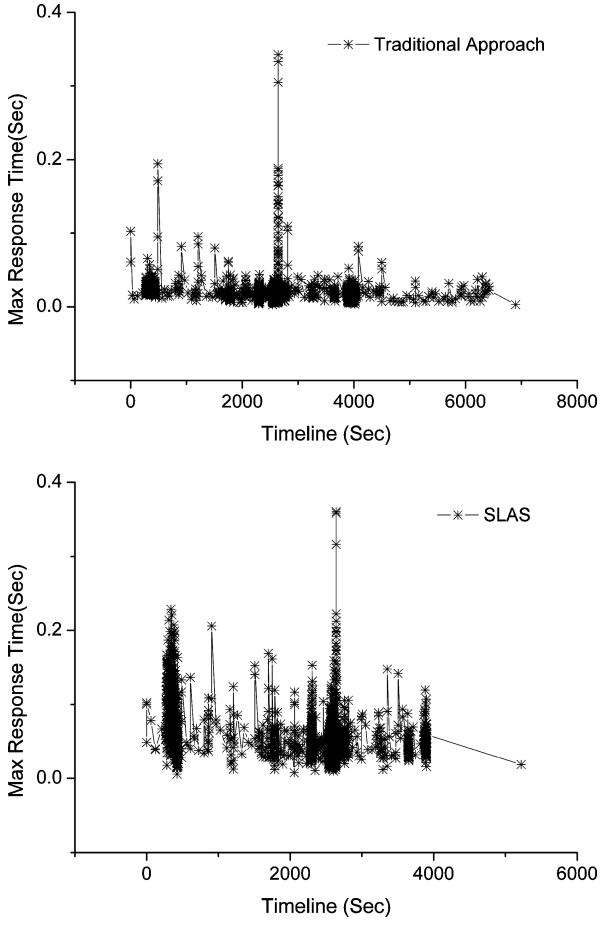

Fig. 14. Local Max-Response-Time series using the traditional and SLAS approaches for A_4^1.

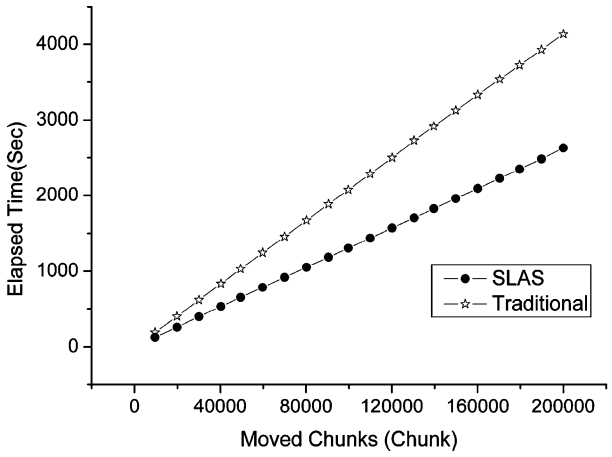

Fig. 15. Elapsed times using the traditional and SLAS approaches for A_2^1.

completion time of the corresponding I/Os for the two approaches. It can be seen that the response time of foreground I/Os is similar. SLAS has a 24.29% shorter redistribution duration than the traditional approach.

The corresponding elapsed time and local max response time series for A_2^1 are shown in Figures 15 and 16. With the shorter response times of foreground I/Os, SLAS has a 36.45% shorter redistribution duration than the traditional approach.

To compare the performance of SLAS with different volume sizes, Figure 17 shows the improvement in redistribution duration by SLAS compared with the traditional approach. All three cases add one disk into a striped volume with the 32KB chunk size under the Cello-92 workload, the only difference among them is that the original volumes are made up of 2, 4, or 8 disks, respectively.

It can be seen that the improvement in redistribution duration by SLAS decreases slightly as the original size of the striped volume increases. This phenomenon can be quite expected: in all our experiments; only one metadata replica on one disk (i.e., Disk 0) was updated during redistribution. If there was no access to Disk 0 since the last metadata update, the current metadata update would have a low cost due to dispensing with a disk seek. When the traditional approach is used, such a chance will increase as the number of disks increases. Under the same workload, therefore, the effect of SLAS's metadata update reduction decreases with an increase of the volume size.

Although the improvement decreases slightly with an increase in the volume size, SLAS makes sense for different volume sizes in most cases for three reasons. First, it can be shown from the previous analysis that when the workload becomes heavier, the decrease of the duration improvement will become slighter. After all, the Cello-92 workload was traced in 1992 and therefore lightly loading. Second, it should also be noted that more than one metadata replica is updated to gain high availability of metadata in most cases. This will cause metadata updates to have a larger share in the cost of data redistribution. Third, large-scale storage systems do not simply use RR-striped volumes. In

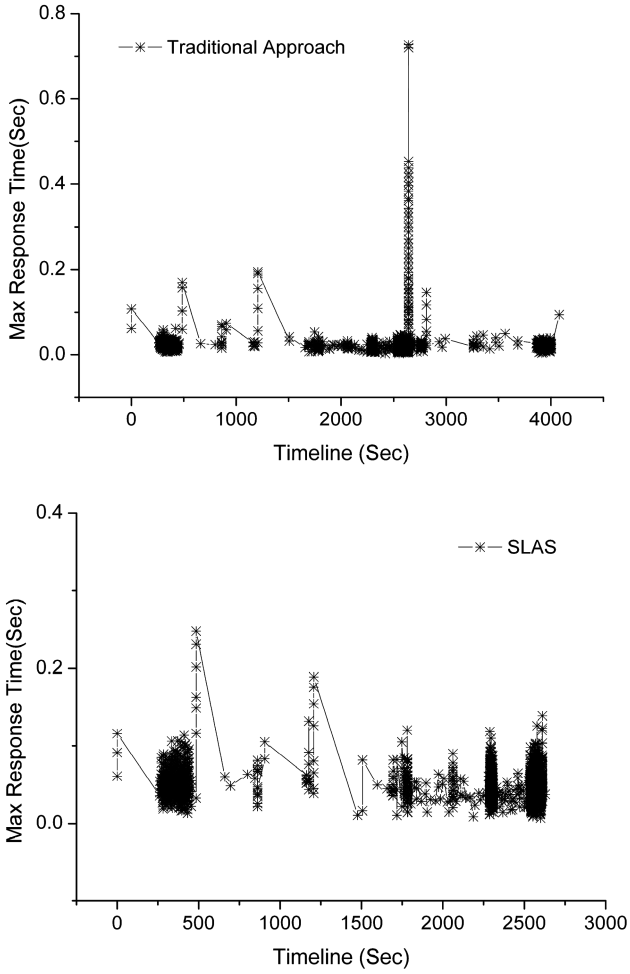

Fig. 16. Local MRT series using the traditional and SLAS approaches for A_2^1.

the implementations of file systems for multimedia, for example, striping is performed over at most a few tens of homogeneous disks [Chou et al. 2000].

6.5 Impact of the Chunk Size

Similarly, we evaluated the effect of changing chunk size on the performance of SLAS. We measured the performance of the traditional and SLAS approaches to perform A_8^1 with the 16KB and 64KB chunk sizes under the Cello-92 workload.

As for the experiment with the 16KB chunk size, the reserved 1.28MB memory for chunk read/write aggregation can hold 80 chunks. Figure 18 shows a plot of the elapsed times versus the numbers of moved chunks using the traditional, SLAS without aggregation, and SLAS approaches. With the similar response time of foreground I/Os, which is not shown for space reasons, SLAS has a 38.04% shorter redistribution duration than the traditional approach. 28.91%

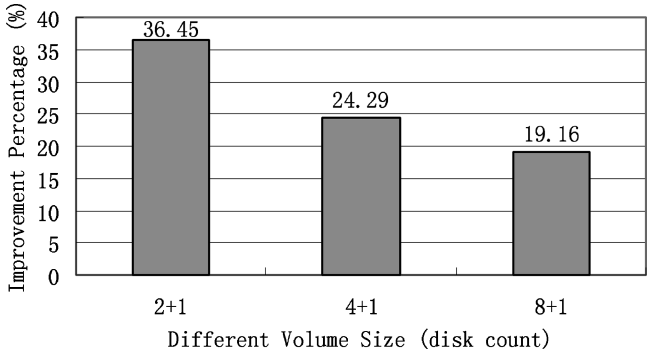

Fig. 17. Effect of varying the volume size.

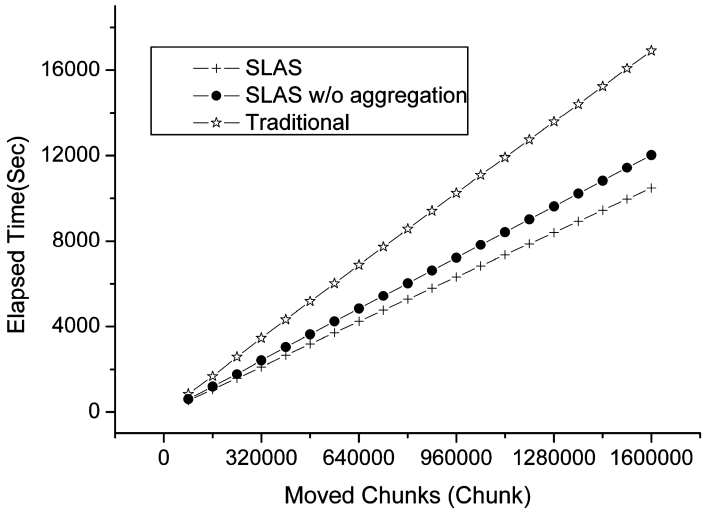

Fig. 18. Elapsed times with the 16KB chunk size.

of this improvement is contributed by lazy updates of mapping metadata, and the other 9.13% is contributed by read/write aggregation of data chunks.

As for the experiment with the 64KB chunk size, the reserved 1MB memory for chunk read/write aggregation can hold 16 chunks. Figure 19 shows a plot of the elapsed times versus the numbers of moved chunks using the traditional, SLAS without aggregation, and SLAS approaches. With the similar response time of foreground I/Os, which is also not shown for space reasons, SLAS has a 7.79% shorter redistribution duration than the traditional approach. 4.55% is contributed by lazy updates of mapping metadata, and the other 3.24% is contributed by read/write aggregation of data chunks.

To compare the performance of SLAS with different chunk sizes, Figure 20 shows the improvement in redistribution duration by SLAS compared with the traditional approach. All the three cases add one disk into a eight-disk striped volume under the Cello-92 workload; the only difference among them is that the chunk sizes are 16KB, 32KB and 64KB and the reserved memory for

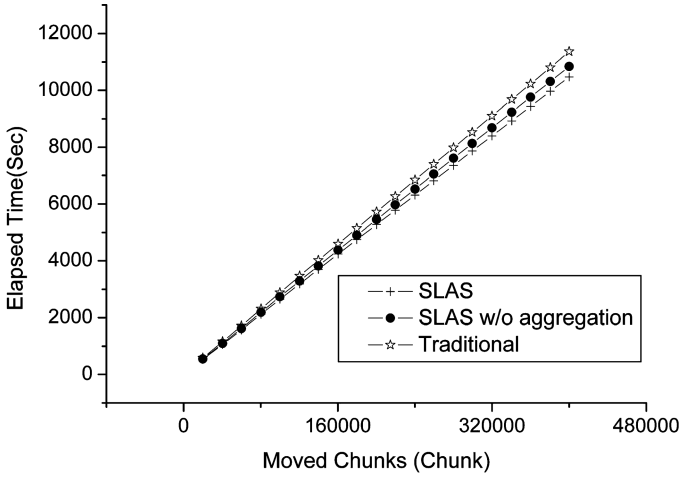

Fig. 19. Elapsed times with the 64KB chunk size.

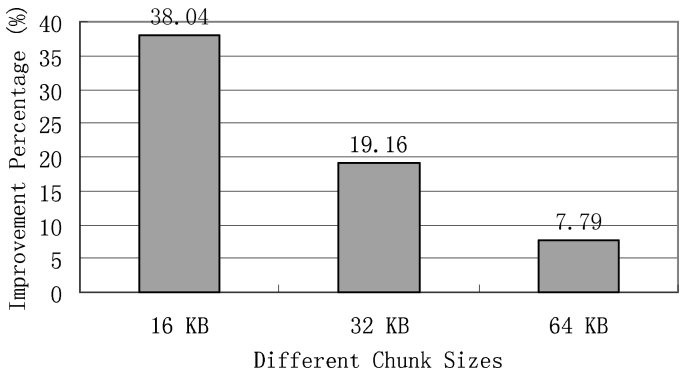

Fig. 20. Effect of varying the chunk size.

read/write aggregation of data chunks can, therefore, hold a different number of data chunks.

It can be seen that the improvement by SLAS decreases as the chunk size increases. We analyze the causes for the decrease as follows. On the one hand, the traditional approach writes mapping metadata onto the disk once for each chunk movement. The cost of metadata updates becomes a smaller portion of the whole redistribution cost as the chunk size increases with the unvaried size of mapping metadata to be updated (512 bytes in our experiments). Therefore, the effect of SLAS's metadata update elimination decreases with an increase of the chunk size. On the other hand, given a fixed sized reserved memory, each aggregative I/O reads/writes about 10, 5, and 2 data chunks, respectively, with the 16KB, 32KB and 64KB chunk sizes. It is also obvious that I/O aggregation has a smaller effect on the bigger reads/writes.

Although the improvement in redistribution duration by SLAS decreases with an increase of the chunk size, it is not advisable to overcome the limitations of the traditional approach by increasing the chunk size. The ultimate

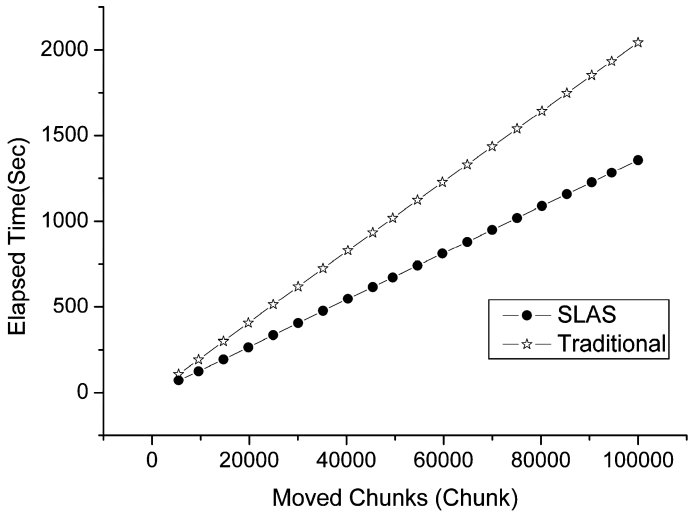

Fig. 21. Elapsed times under the TPC-C workload.

goal of creating an RR-striped volume is not to scale it quickly but to improve the I/O performance of a storage system. The chunk size must be chosen according to the workload characteristics. A too large chunk size will make multiple disks unable to serve I/O requests with an adequate parallelism. In addition, the track size in our experiments is as small as 123.5–195KB. As the track size of disks grows over time (200–350KB for 2002 disks) [Schindler et al. 2004], read/write aggregation in SLAS will make sense for increasingly larger chunk sizes. The reason is that aggregate reads/writes do not require disk seeks except when the disk head moves from the final block of one track to the first block of another track.

6.6 Impact of the Foreground Workload

Another factor that might affect the benefits of SLAS is the workload under which data redistribution is performed. We also measured the performance of the traditional and SLAS approaches to perform A_2^1 with the 32KB chunk size under the TPC-C and Cello-96 workloads.

As far as the TPC-C workload is concerned, Figure 21 plots the elapsed times versus the number of data chunks that are already moved using the two approaches as the data redistribution proceeded, and Figure 22 shows a plot of 2,000 local MRTs versus the completion times of the corresponding I/Os using the two approaches. Under the TPC-C workload, with the similar response time of foreground I/Os, SLAS shows an improvement of 33.60% in redistribution duration.

The corresponding elapsed time and local max response time series under the Cello-96 workload are shown in Figure 23 and Figure 24. With the similar response time of foreground I/Os, SLAS has a 40.79% shorter redistribution duration than the traditional approach.

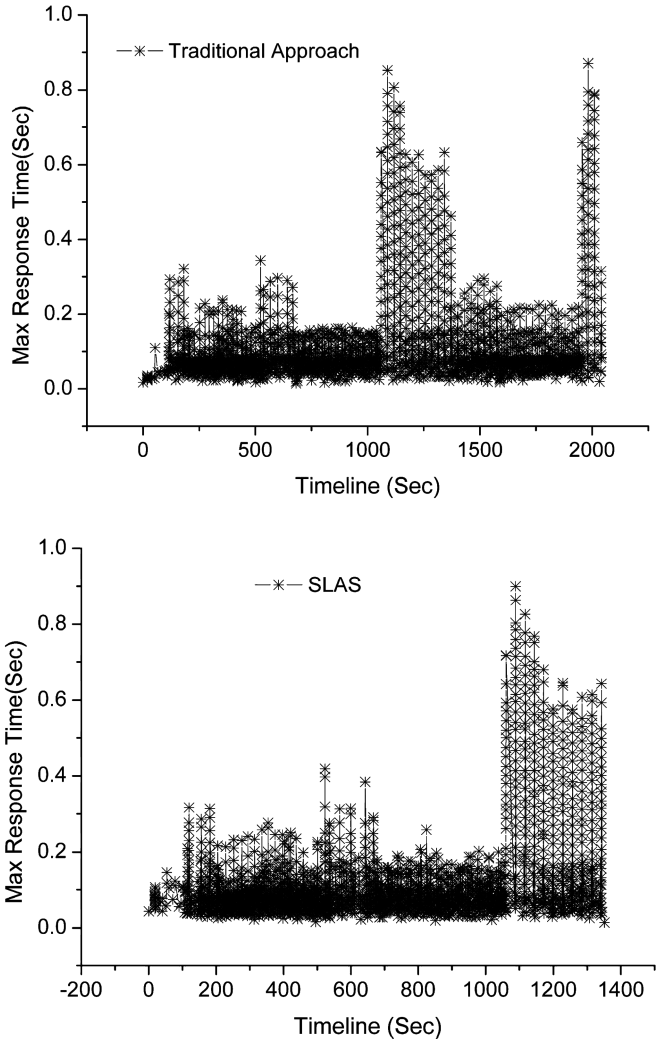

Fig. 22. Local MRT series under the TPC-C workload.

The bar graph in Figure 25 compares the improvements in redistribution duration by SLAS under the Cello-92, TPC-C and Cello-96 workloads. We can see that the improvement varies when the workload is different, but the variation is slight.

7. DISCUSSION ABOUT THE EXTENDABILITY OF SLAS

Widely-used RR-striped volumes can be categorized into three groups: RAID-0, RAID-3, and RAID-5. We mainly discuss the issue of adding disks into a RAID-0 volume. This section will analyze such issues as removing disks from a RAID-0 volume and adding/removing disks into/from a RAID-4 or RAID-5 volume.

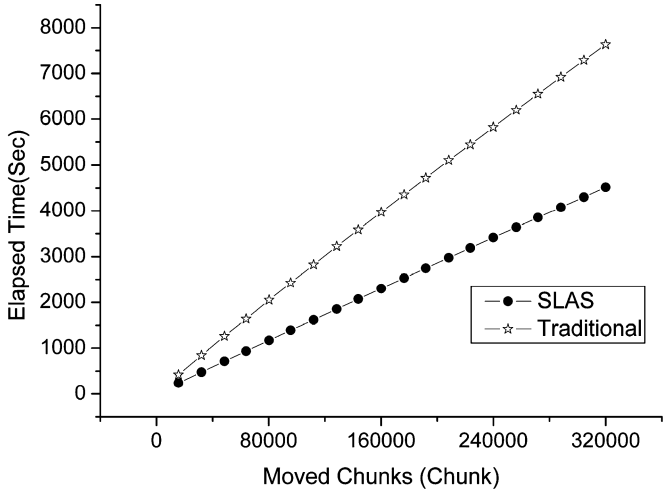

Fig. 23. Elapsed times under the Cello-96 workload.

7.1 Removal of Disks from RAID-0 Volumes

Similar to adding disks into a RAID-0 volume, disk removal also necessitates the data on the volume to be reorganized. The only difference is that disk removal performs data redistribution from the end to the beginning of the volume while disk addition does it in the opposite order. Therefore, if we can prove that there is also a reordering window during redistribution for disk removal, our SLAS approach can be used in disk removal with a little revision. In fact, such a reordering window does exist.

Symmetric to disk addition, we give some basic definitions for disk removal: R_n^m, $S(n, m, x)$, and ROW(n, m, x). Here, R_n^m denotes a request to remove m disks from an n-disk volume. To clarify the existence of a reordering window, we take $S(3, 1, 8)$ as an example. As shown in Figure 26, when Chunks 6, 7, and 8 are moved in arbitrary order, no valid chunk will be overwritten. If Chunk 5 is also taken into account, when Chunk 5 is moved before Chunk 7, the former will overwrite the latter. So we get ROW(3,1,8) = 3.

Generally speaking, we can calculate the size of a reordering window for disk removal as Lemma 2.

LEMMA 2. $$ROW(n, m, x) = \begin{cases} m \times \lfloor x \ / \ n \rfloor, if\, x \,\%\, n \ < n - m. \\ m \times \lfloor x \ / \ n \rfloor + x\%n - (n - m) + 1, else. \end{cases}.$$

PROOF. As for $S(n, m, x)$, let $OD(x) = d$, $OE(x) = e$, where both d and e start at 0. We have $e = \lfloor x/n \rfloor$, $d = x\%n$. We shall break it into two cases: (1) disk d will not be removed from the volume; (2) disk d will be removed.

(1) If disk d is not removed, then d = x% n < n − m.

In this case, another logical chunk x' will be placed on Chunk e of Disk d in the new configuration. Thus $ND(x') = d$, $NE(x') = e$, then $x' = e \times (n-m) + d = \lfloor x/n \rfloor \times (n-m) + (x\%n)$. Further, we have $\delta = x - x' = x - (\lfloor x/n \rfloor \times (n-m) + x\%n) = m \times \lfloor$x/n$\rfloor$. In the following, we prove that δ is just the size of the reordering window of R_n^m at Chunk x by contradiction.

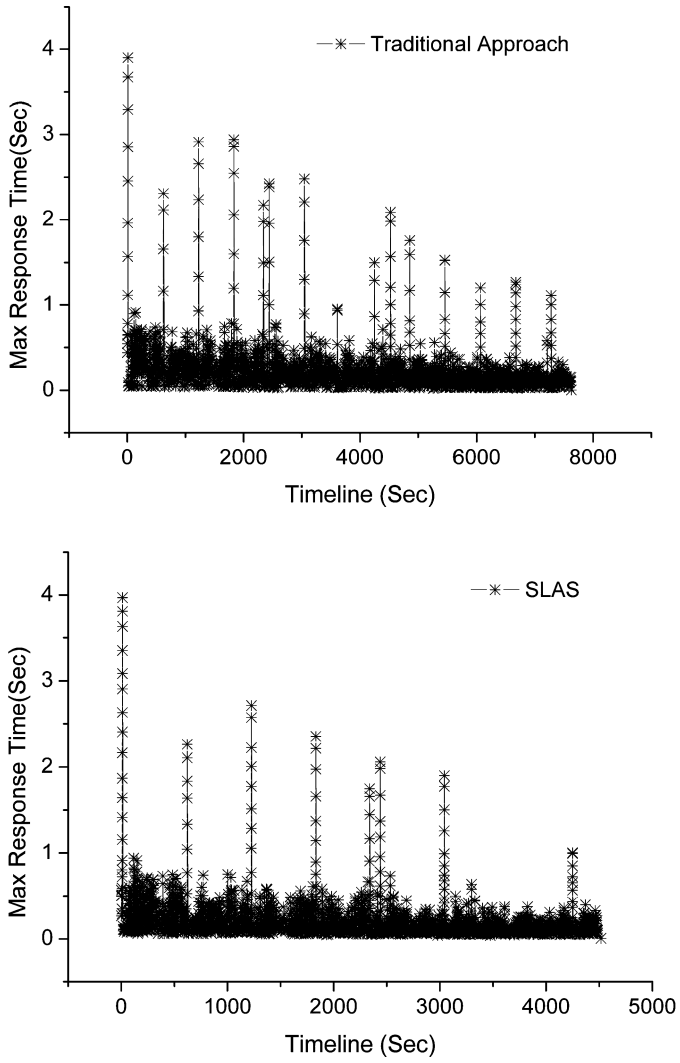

Fig. 24. Local MRT series under the Cello-96 workload.

First, we assume that, $\exists\, x_a\, \exists\, x_b\{x_a, x_b \in (x-\delta, x]$ and $x_a < x_b\}$ when other.}. data chunks x_a, x_b are moved in some order, one will be overwritten by the other.}. It is evident that no valid chunk will be overwritten if Chunk x_b is moved first. So, Chunk x_a is moved first and overwrites Chunk x_b. Therefore, $x_a = \lfloor x_b/n \rfloor \times (n-m) + (x_b \% n)$. Since $x_b \leq x$, we have $x_a \leq x' = x - \delta$. This result contradicts our initial assumption that $x_a \in (x-\delta, x]$. Therefore, for $\forall x_a \forall x_b\{x_a, x_b \in (x-\delta, x]$ and $x_a < x_b\}$, data chunks x_a and x_b cannot be overwritten by each other no matter in what order they are moved.

Additionally, for arbitrary nonnegative integer $\delta'\{\delta' > \delta\}$, we have $x, x' \in (x-\delta', x]$. If Chunk x' is moved before Chunk x, it will overwrite Chunk x.

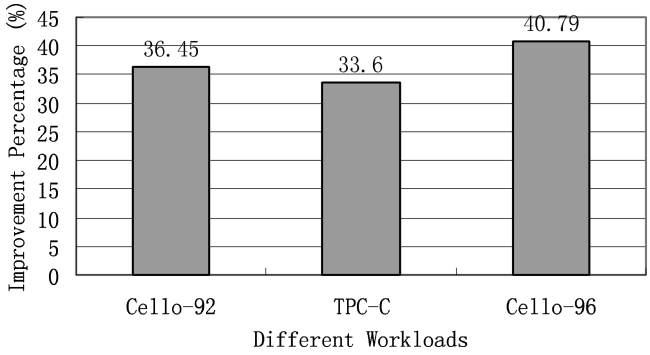

Fig. 25. Effect of varying the workload.

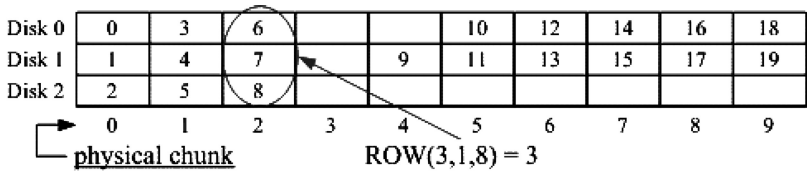

Fig. 26. Illustration of S(3,1,8). (Disk 2 is removed.)

According to the definition of a reordering window, δ is the size of the reordering window of R_n^m at Chunk x. That is, if x% n < n-m, then ROW(n, m, x) = δ = m × ⌊x / n⌋.

(2) If disk d is removed, then d = x% n ≥ n − m.

In this case, no logical chunk will be placed on Chunk e of Disk d in the new configuration. Let us assume that $x_1 = \underset{y}{MAX}\{$y | y is a nonnegative integer; $y < x$, and Disk $OD(y)$ is not removed.$\}$, we have $OD(x_1) = n - m - 1$, $OE(x_1) = e$. Another logical chunk x' will be placed on Chunk e of Disk $n - m - 1$ in the new configuration. Thus $ND(x') = n - m - 1$, $NE(x') = e$, and then $x' = e \times (n - m) + n - m - 1 = \lfloor x/n \rfloor \times (n - m) + n - m - 1$. Further, we have $\delta = x - x' = x - (\lfloor x/n \rfloor \times (n - m) + n - m - 1) = m \times \lfloor$x / n$\rfloor + x\%n - (n - m) + 1$. In the following, we prove that δ is just the size of the reordering window of R_n^m at Chunk x by contradiction.

First, we assume that, $\exists\, x_a\, \exists\, x_b\{x_a, x_b \in (x - \delta, x]$ and $x_a < x_b\}$. When other.$\}$. data chunks x_a, x_b are moved in some order, one will be overwritten by the other.$\}$. It is evident that no valid chunk will be overwritten if Chunk x_b is moved first. So, Chunk x_a is moved first and overwrites Chunk x_b. Therefore, $x_a = \lfloor x_b/n \rfloor \times (n - m) + (x_b\%n)$ and Disk OD(x_b) is not removed.

Note that Disk $OD(x_b)$ is not removed, since $x_b \leq x$, we have $x_b \leq$ x$_1$. Further, we have $x_a \leq x' = x - \delta$. This result contradicts our initial assumption that $x_a \in (x - \delta, x]$. Therefore, for $\forall x_a \forall x_b\{x_a, x_b \in (x - \delta, x]$ and $x_a < x_b\}$, data chunks x_a and x_b cannot be overwritten by each other no matter in what order they are moved.

Additionally, for arbitrary nonnegative integer $\delta'\{\delta' > \delta\}$, we have $x_1, x' \in (x - \delta', x]$. If Chunk x' is moved before Chunk x_1, it will overwrite Chunk x_1.

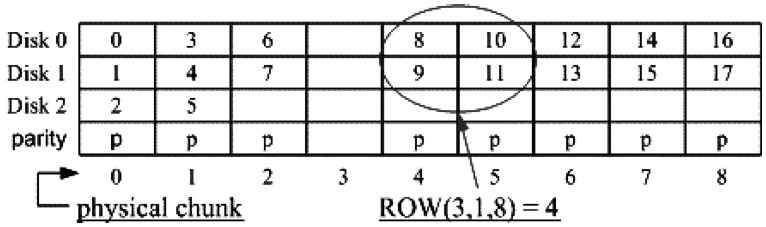

Fig. 27. Illustration of S(3,1,8) for RAID-4. (Disk 2 is newly added, 'p' stands for a parity chunk.)

According to the definition of a reordering window, δ is the size of the reordering window of R_n^m at Chunk x. That is, if x% n $\geq$ n $-$ m, ROW(n, m, x) $=$ δ = m $\times$ $\lfloor$x/n$\rfloor$ + $x\%n - (n - m) + 1$.

Summarize these two cases, Lemma 2 is proved. □

Now that a reordering window exists during redistribution for disk removal, our SLAS approach can be used in disk removal from a RAID-0 volume with a little revision. Thus, our SLAS approach works for both disk addition and disk removal to/from a RAID-0 volume effectively.

7.2 Scaling RAID-4 and RAID-5 Volumes

RAID-0 boosts performance best but does not provide data security. Therefore, some other RR-striped volumes like RAID-4 and RAID-5 are also used widely. Here, we discuss whether our SLAS approach can be extended for scaling a RAID-4 or RAID-5 volume. To achieve this goal, we need to make clear whether there is also a reordering window during redistribution for scaling a RAID-4 or RAID-5 volume.

A RAID-4 volume is identical to a RAID-0 volume except that a parity disk is added to hold all parity chunks for the whole volume. Figure 27 illustrates the disk-scaling state S(3, 1, 8) when one disk is added into a RAID-4 volume. We can see that ROW(3, 1, 8) = 4. Similar to scaling a RAID-0 volume, there is also a reordering window during redistribution for scaling a RAID-4 volume. We can calculate the size of a reordering window for adding/removing disks to/from a RAID-4 volume, respectively, by Lemma 3 and Lemma 4, which are easily gained imitating the proofs of Lemma 1 and Lemma 2.

LEMMA 3. $ROW(n, m, x) = m \times \lfloor x \ / \ (n - 1) \rfloor$.

LEMMA 4. $ROW(n, m, x) =$

$$\begin{cases} m \times \lfloor x/(n-1) \rfloor, if\, x\%(n-1) < (n\text{ - }1) - m. \\ m \times \lfloor x/(n-1) \rfloor + x\%(n-1) - (n-1-m) + 1, else. \end{cases}$$

Every write to a RAID-4 volume has to write new parity chunks to the parity disk, so the parity disk is heavily loaded while data disks are relatively idle. This will bring a negative impact on the performance of RAID-4. As an improvement on RAID-4, RAID-5 distributes parity chunks across all the disks in a round-robin fashion.

To demonstrate how to calculate the size of a sliding window for disk addition in a RAID-5 volume, we take A_3^1 as an example. Figure 28

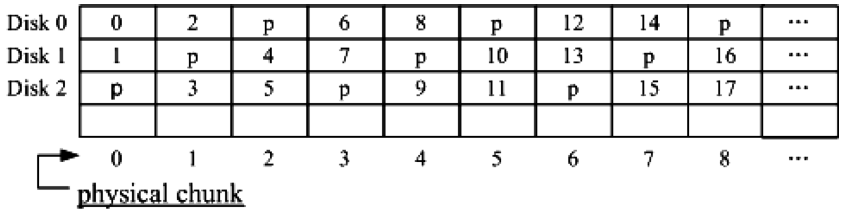

Fig. 28. Data organization for a RAID-5 volume before adding a disk. ('p' stands for a parity chunk.)

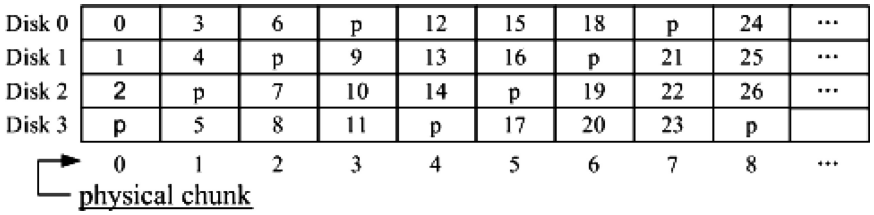

Fig. 29. Data organization for a RAID-5 volume after adding a disk. (Disk 3 is newly added, 'p' stands for a parity chunk.)

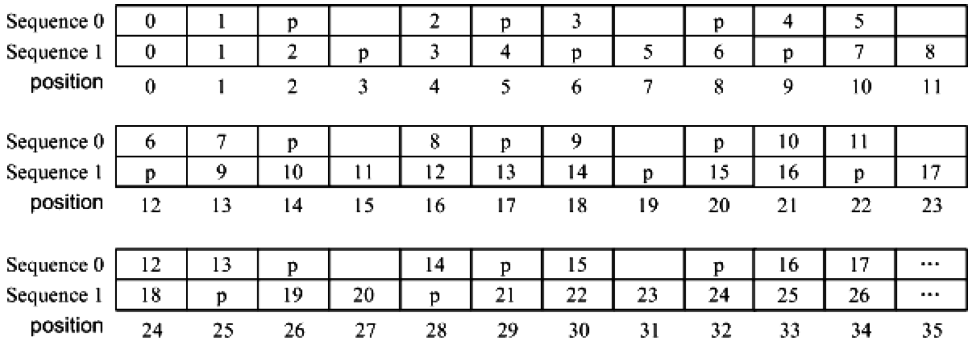

Fig. 30. Sequence representation of data organizations before and after adding a disk. ('p' stands for a parity chunk.)

and Figure 29 illustrate data organizations in the original and new configurations.

We can represent the two data organizations with two sequences as shown in Figure 30, where sequence 0 represents the organization in the original configuration, and sequence 1 represents the one in the new configuration. In fact, to redistribute data is to overwrite the data of sequence 0 with the data of sequence 1. We shall break it into four cases: (1) a data chunk overwrites a data chunk; (2) a data chunk overwrites a parity chunk; (3) a parity chunk overwrites a data chunk; (4) a parity chunk overwrites a parity chunk. According to the concept of a reordering window, to calculate $ROW(n, m, x)$, we need to take cases (1) and (3) into account.

Before disk addition, for arbitrary chunk x, its possition $p(x)$ in sequence 0 can be calculated as follows:

We let $\mathrm{q} = \lfloor \mathrm{x}/(\mathrm{n} \times (\mathrm{n}-1)) \rfloor$ and $\mathrm{r} = \mathrm{x}\%\ (\mathrm{n} \times (\mathrm{n}-1))$.

(a) When $\mathrm{r}\%\ (\mathrm{n}-1)+1 \geq \mathrm{n} - \lfloor \mathrm{r}/(\mathrm{n}-1) \rfloor$,

$$\mathrm{p}(x) = q \times (n^2 + n \times m) + \lfloor \mathrm{r}/(\mathrm{n}-1) \rfloor \times (n+m) + r\%(n-1) + 1;$$

(b) When $\mathrm{r}\%\ (\mathrm{n}-1)+1 < \mathrm{n} - \lfloor \mathrm{r}/(\mathrm{n}-1) \rfloor$,

$$\mathrm{p}(x) = q \times (n^2 + n \times m) + \lfloor \mathrm{r}/(\mathrm{n}-1) \rfloor \times (n+m) + r\%(n-1).$$

If we know the position $p(x)$ of chunk x in sequence 0, we can also calculate the chunk x′ in the same position in sequence 1 as follows:

We let $\mathrm{q}' = \lfloor \mathrm{p(x)} / (\mathrm{n}+\mathrm{m}) \rfloor$ and $\mathrm{r}' = p(\mathrm{x})\%\ (\mathrm{n}+\mathrm{m})$.

(c) When $\mathrm{r}' + 1 > \mathrm{n} + \mathrm{m} - \mathrm{q}'\ \%\ (\mathrm{n}+\mathrm{m})$,

$$\mathrm{x}' = \mathrm{q}' \times (\mathrm{n}+\mathrm{m}-1) + \mathrm{r}' - 1;$$

(d) When $\mathrm{r}' + 1 \leq \mathrm{n} + \mathrm{m} - \mathrm{q}'\ \%\ (\mathrm{n}+\mathrm{m})$,

$$\mathrm{x}' = \mathrm{q}' \times (\mathrm{n}+\mathrm{m}-1) + \mathrm{r}'.$$

Finally, $x' - x$ is just the size of the sliding window. That is, $ROW(n, m, x) = x' - x$.

As far as disk addition in a RAID-5 volume is concerned, we do not present the strict and detailed proof for space reasons. But we can see that there does exist a reordering window during adding disks to a RAID-5 volume, and we have the ability to calculate its size. The only difference between disk removal and disk addition is that disk removal performs data redistribution in the opposite order. Therefore we get a conclusion that there is also a reordering window during removing disks from a RAID-5 volume.

Now that a reordering window exists during redistribution for disk addition/removal in a RAID-4 or RAID-5 volume, our SLAS approach can be extended for scaling a RAID-4 or RAID-5 volume with some revisions.

8. CONCLUSIONS AND FUTURE WORK

The contributions of this article are twofold. First, we present the concept of a reordering window, which provides a theoretical basis for solving the problem of scaling RR-striped volumes (including RAID-0, RAID-4, and RAID-5). Second, taking advantage of the reordering window characteristic, we propose SLAS, an efficient approach to scaling RR-striped volumes.

During data redistribution caused by scaling RR-striped volumes, there is always a reordering window where data consistency can be maintained while changing the order of data movements. Some conclusions drawn from the concept of a reordering window are very important to solving the problem of scaling striped volumes.

The SLAS approach uses a new mapping-management solution based on a sliding window to support data redistribution without loss of scalability. Moreover, there are two main ideas in SLAS to improve the redistributing efficiency.

The first idea is to use lazy updates of mapping metadata to decrease the number of metadata writes required by data redistribution. The second idea is to change the order of chunk movements to aggregate reads/writes of data chunks.

Compared with the traditional approach, our SLAS approach similarly guarantees data consistency and does not enlarge the impact on the response time of foreground I/Os. However, SLAS has the ability to shorten the redistribution duration markedly. Our results from detailed simulations using real-system traces indicate some conclusions as follows.

(1) SLAS can shorten the redistribution duration by up to 40.79% with similar maximum response time of foreground I/Os.
(2) Both lazy updates of mapping metadata and chunk read/write aggregation make a large contribution to the improvement.
(3) The improvement in redistribution duration by SLAS decreases slightly as the number of the original disks in the striped volume increases.
(4) The improvement by SLAS decreases as the chunk size increases.
(5) The improvement varies when workloads are different, but the variation is quite slight.

Finally, our discussion indicates that SLAS can not only be used in adding new disks to a RAID-0 volume; it can also be extended to remove existing disks from a RAID-0 volume and to add/remove disks to/from in a RAID-4 or RAID-5 volume.

One aspect of scaling RR-striped volumes that we have not addressed in this article is how to ensure that foreground I/Os are not impacted significantly while data redistribution is in progress. At present, the SLAS approach moves data online in a best-effort manner as the traditional approach does, which could bring an unpredictable performance influence to foreground I/Os in loaded systems. In the future, we will do research on rate control in scaling RR-striped volumes.

ACKNOWLEDGMENTS

The authors would like to thank Professor Greg Ganger at Carnegie Mellon University who gave us some good advices in using the disksim simulator. We appreciate Professor Nianmin Yao, our former colleague at Tsinghua University who helped us in Simpy programming. We are also grateful to Professor Wenguang Chen at Tsinghua University for the constructive discussion and thank HP storage system labs for providing us real traces. Special thanks to the anonymous reviewers for their invaluable feedback.

REFERENCES

ALEMANY, J. AND THATHACHAR, J. S. 1997. Random striping news on demand servers. Tech. rep. TR-97-02-02, University of Washington.

BERSON, S., GHANDEHARIZADEH, S., MUNTZ, R., AND JU, X. 1994. Staggered striping in multimedia information systems. SIGMOD. 79–90.

BUCY, J. S. AND GANGER, G. R. 2003. The DiskSim Simulation Environment Version 3.0 Reference Manual. Tech. rep. CMU-CS-03-102, Carnegie Mellon University.

CHEN, P. AND PATTERSON, D. 1990. Maximizing performance in a striped disk array. In *Proceedings of ACM SIGARCH Conference on Computer Architecture*. Seattle, WA, 322–331.

CHOU, C. F., GOLUBCHIK, L., AND LUI, J. C. S. 2000. Striping doesn't scale: How to achieve scalability for continuous media servers with replication. In *Proceedings of IEEE ICDCS*. 64–71.

DASGUPTA, K., GHOSAL, S., JAIN, R., ET AL. 2005. QoSMig: Adaptive rate-controlled migration of bulk data in storage systems. In *Proceedings of the International Conference on Data Engineering (ICDE'05)*. 816–827.

GHANDEHARIZADEH, S. AND KIM, D. 1996. Online reorganization of data in scalable continuous media servers. In *Proceedings of the 7th International Conference on Database and Expert Systems Applications*. Zurich, Switzerland. Lecture Notes in Computer Science, DG. Feitelson and L. Rudolph, Eds. 751–768.

GOEL, A., SHAHABI, C., YAO, S. Y., AND ZIMMERMANN, R. 2002. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*. Chaudhuri S., Carey M., and Garcia-Molina H., Eds. San Jose, IEEE CS Press, 473–482.

GONZALEZ, J. L. AND CORTES, T. 2004. Increasing the capacity of RAID5 by online gradual assimilation. *International Workshop on Storage Network Architecture and Parallel I/Os.* Antibes Juan-les-pins, France.

HENNESSY, J. L. AND PATTERSON, D. A. 2003. *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

KIM, C. S., KIM, G. B., AND SHIN, B. J. 2001. Volume management in SAN environment. In *Proceedings of the 8th International Conference on Parallel and Distributed Systems (ICPADS '01)*. 500–505.

LEWIS, A. J. 2005. LVM HOWTO. http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/ LVM-HOWTO.pdf.

LIM, S. H., HWANG, J. Y., KIM, K. H., ET AL. 2003. Resource volume management for shared file system in SAN environment. In *Proceedings of the 16th International Conference on Parallel and Distributed Computing Systems (PDCS)*.

LIVNY, M., KHOSHAFIAN, S., AND BORAL, H. 1987. Multi-disk management algorithms. *ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*. 69–77.

LU, C., ALVAREZ, G. A., AND WILKES, J. 2002. Aqueduct: Online data migration with performance guarantees. In *Proceedings of the 1st USENIX Conference on File and Storage Technologies (FAST'02)*. 219–230.

RUEMMLER, C. AND WILKES, J. 1993. A trace-driven analysis of disk working set sizes. Tech. rep. HPL–OSR–93–23, Hewlett-Packard Laboratories, Palo Alto, CA.

SCHINDLER, J., SCHLOSSER, S. W., ET AL. 2004. Atropos: A disk array volume manager for orchestrated use of disks. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04)*. San Francisco, CA.

SEO, B. AND ZIMMERMANN, R. 2005. Efficient disk replacement and data migration algorithms for large disk subsystems. *ACM Trans. Storage 1*, 3, 316–345.

STERGAARD, J. 2001. RAID Reconfiguration Tool. http://unthought.net/raidreconf/

TEIGLAND, D. AND MAUELSHAGEN, H. 2001. Volume managers in linux. In *Proceedings of the 2001 USENIX Annual Technical Conference*. 185–198.

VERMA, A., SHARMA, U., RUBAS, J., ET AL. 2005. An architecture for lifecycle management in very large file systems. In *Proceeding of the 22nd IEEE-13th NASA Goddard Conference on Mass Storage Systems and Technology (MSST'05)*.

VIGNAUX, T. AND MULLER, K. 2005. SimPy Manual. http://simpy.sourceforge.net/SimPyDocs/Manual.html. Nov. 2005.

VOGELS, W. 1999. File system usage in Windows NT 4.0. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*. 93–109.

WILKES, J., GOLDING, R., STAELIN, C., AND SULLIVAN, T. 1996. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst. 14*, 1, 108–136.

XIAO, D., SHU, J.W., XUE, W., AND ZHENG, W. M. 2005. TH-VSS: An asymmetric storage virtualization system for the SAN environment. In *Proceedings of the International Conference on Computational Science*. 399–406.

YU, X., GUM, B., CHEN, Y., ET AL. 2000. Trading capacity for performance in a disk array. In *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*.

ZHANG, G. Y., SHU, J. W., XUE, W., AND ZHENG, W. M. 2005. MagicStore: A new out-of-band virtualization system in SAN environments. In *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC'05)*. Nov. Lecture Notes in Computer Science, vol. 3779, 379–386.

Received May 2006; revised January 2007; accepted January 2007