ALV: A New Data Redistribution Approach to RAID-5 Scaling

Guangyan Zhang, Weimin Zheng, and Jiwu Shu

Abstract—When a RAID-5 Volume is scaled up with added disks, data have to be redistributed from original disks to all disks including the original and the new. Existing online scaling techniques suffer from long redistribution time as well as negative impact on application performance. By leveraging our insight into a reordering window, this paper presents ALV, a new data redistribution approach to RAID-5 scaling. The reordering window is a result of the natural space hole as data being redistributed, and it grows in size. The data inside the reordering window can migrate in any order without overwriting other in-use data chunks. The ALV approach exploits three novel techniques. First, ALV changes the movement order of data chunks to access multiple successive chunks via a single I/O. Second, ALV updates mapping metadata lazily to minimize the number of metadata writes while ensuring data consistency. Third, ALV uses an on/off logical valve to adaptively adjust the redistribution rate depending on application workload. We implemented ALV in Linux Kernel 2.6.18 and evaluated its performance by replaying three real-system traces: TPC-C, Cello-99, and SPC-Web. The results demonstrated that ALV outperformed the conventional approach consistently by 53.31-73.91 percent in user response time and by 24.07-29.27 percent in redistribution time.

Index Terms—RAID-5 scaling, reordering window, I/O aggregation, lazy checkpoint, rate control.

1 Introduction

Due to I/O parallelism and high availability, RAID-5 is widely used on servers. Disk addition to a RAID-5 volume can meet two requirements simultaneously: larger storage capacity and higher I/O bandwidth. This disk addition is termed *"RAID-5 scaling."*

Performing the process of RAID-5 scaling is a difficult technical challenge for two reasons. First, almost every data chunk has to be moved to preserve the round-robin order. Second, due to the extremely high cost of downtime [1], the data redistribution has to be performed online. Scaling a RAID-5 volume requires an efficient approach to redistributing the data online, so that: 1) data redistribution will be completed in a short time; 2) the impact of data redistribution on application performance will not be significant; and 3) data consistency will be guaranteed even if the system crashes or one disk fails during the scaling process.

So far, several approaches have been proposed for redistributing data for RAID-5 scaling. Typical examples include the gradual assimilation (GA) algorithm [2] and the reshape toolkit in a Linux MD driver (MD-Reshape) [3]. Both of them suffer from a low data redistribution efficiency due to the following limitations: 1) *Issuing small redistribution I/Os.* Data redistribution has a sequential access pattern, which is interleaved with user requests. Issuing small redistribution I/Os breaks up this sequential streaming, hence, bringing down the redistribution efficiency by a certain magnitude [4]. 2) *Updating mapping metadata frequently.* In order to keep data consistent, these approaches frequently write mapping metadata onto disks (a.k.a., *checkpoint*). Because metadata are usually stored at the beginning of all member disks, each metadata update causes one long seek per disk. 3) *Using nonadaptive rate control.* Existing approaches use nonadaptive schemes to control redistribution rate so as to avoid interfering with application performance to unacceptable levels. Due to workload fluctuations on arbitrary time scales [5], these nonadaptive schemes either degrade application performance greatly or result in unsatisfactory redistribution speeds.

Data redistribution requires 1) reading and writing the data chunks and 2) updating the mapping metadata to record data movements. It is impossible to reduce the number of data chunks moved while still maintaining the round-robin order. Consequently, we focused on decreasing the numbers of data accesses and metadata updates.

If data chunks have to be moved in a strict one by one sequence, it will be difficult to optimize the data redistribution process. Our previous research [6] has discovered that there is always a reordering window during data redistribution for RAID-0 scaling. The data inside the reordering window can migrate in any order without overwriting any valid data. By leveraging our insight into a reordering window, SLAS is proposed to scale RAID-0 volumes online, which reduces the cost of data redistribution effectively.

We can predict that there is also a reordering window during RAID-5 scaling. However, optimizing data redistribution for RAID-5 scaling will be more difficult in order to maintain data parity. First, for efficient data redistribution, calculating the size of the reordering window is required, which will be more difficult. Second, write operations are indispensable for data redistribution. The performance difference between partial-stripe writes and full-stripe writes is significant [7], [8]. For workloads of mostly small writes, the throughput of RAID-5 arrays is

• *The authors are with the Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Room 8-208, East Main Building, Beijing 100084, P.R. China. E-mail: {gyzh, zwm-dcs, shujw}@tsinghua.edu.cn.*

Manuscript received 1 Nov. 2008; revised 6 June 2009; accepted 16 July 2009; published online 25 Sept. 2009.
Recommended for acceptance by M. Gokhale.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-11-0542.
Digital Object Identifier no. 10.1109/TC.2009.150.

penalized by a factor of 4 over RAID-0 arrays. SLAS, designed and optimized for RAID-0 scaling, does not take care of this difference. As a result, the small write problem may arise in redistribution operations. What's more, SLAS has an important limitation—it moves data in a best-effort manner. This will have a significant impact on application performance in heavily loaded systems.

In this paper, we first quantify how the reordering window grows in size as data redistribution progresses. Then, we propose ALV,[1] a new data redistribution approach to RAID-5 scaling. It makes use of three key techniques as follows:

- ALV changes the order of data movements to access multiple successive chunks via a single I/O. As a result, ALV requires fewer but larger redistribution I/Os. The sequential pattern of data redistribution is interleaved with user requests, so larger redistribution I/Os amortize positioning delays over more data transfers, increasing redistribution throughput [4]. Also, ALV uses *the write alignment technique* to eliminate extra reads to old data and old parity for calculating new parity. ALV moves the data once, whose size is exactly a multiple of a stripe in the new geometry.
- ALV updates mapping metadata lazily to minimize the number of metadata writes. Data movement is not checkpointed until a threat to data consistency occurs. In this way, the number of metadata writes is reduced significantly. Moreover, even if the system or one disk fails unexpectedly, only some data accesses will be wasted, while data remain consistent.
- ALV uses an on/off logical valve to adaptively adjust the redistribution rate depending on application workload. Data redistribution is throttled on detection of high application workload. Otherwise, it performs continuously. The goal is to maximize redistribution rate dynamically without a significant impact on application performance.

We implemented the ALV approach in the software RAID of Linux Kernel 2.6.18 instead of a simulation system, which is used to evaluating SLAS. The benchmark studies on the three real-system workloads (i.e., TPC-C, Cello-99, and SPC-Web) showed that ALV outperformed MD-Reshape by 53.31-73.91 percent in user response time and by 24.07-29.27 percent in redistribution time simultaneously.

2 An Insight into a Reordering Window

We discover that there is always a reordering window during data redistribution for RAID-5 scaling. To understand how the reordering window arises and grows in size as the redistribution progresses, we take RAID-5 scaling from 3 disks to 4 as an example. Fig. 1 shows the sequence of the first six reordering windows.[2] Only chunk 2 can first be moved once, then only chunk 3, then both 4 and 5, then 6-8, then 9-12, and so on. Let us focus on state 3. When Chunks 6, 7, and 8 are moved in an arbitrary order, no valid chunk is overwritten. If Chunk 9 is also taken into account, when Chunk 9 is moved before Chunk 7, the former overwrites the latter. Here, Chunks 6, 7, and 8 make up *a reordering window*.

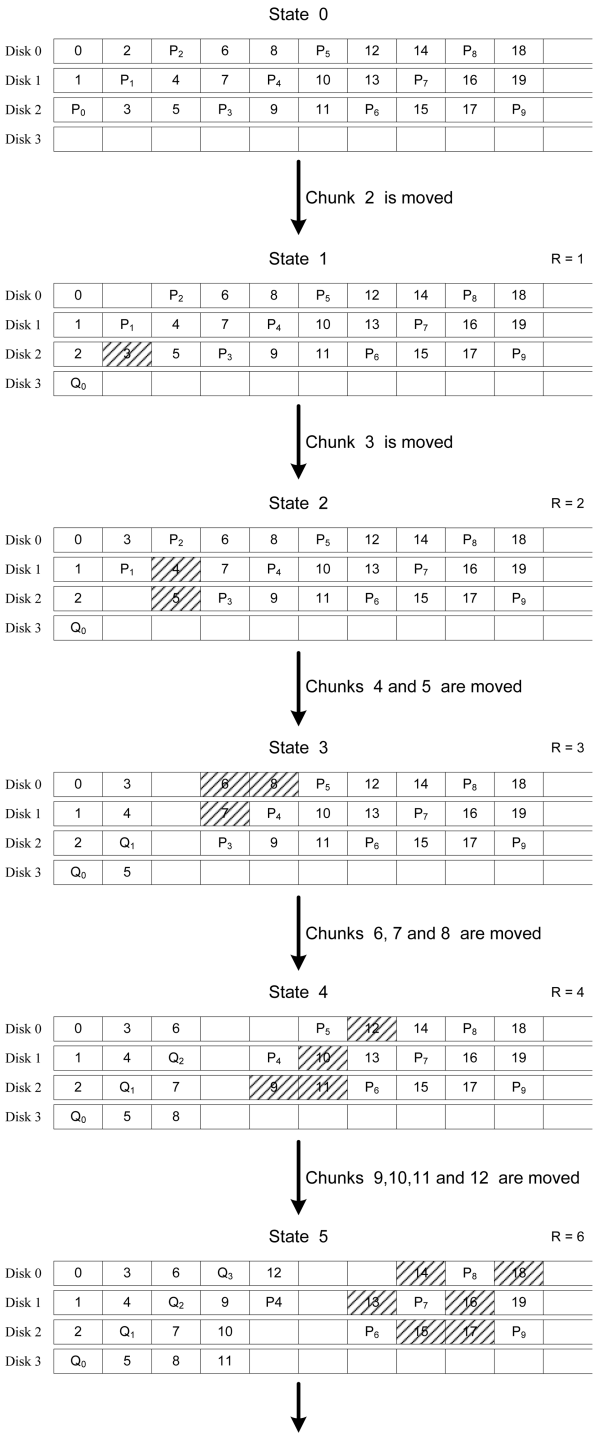

Fig. 1. A series of states in data redistribution for RAID-5 scaling from 3 disks to 4. The size of the reordering window, represented by "R", increases gradually.

We give three definitions about a reordering window as follows:

Definition 1. *Given a RAID-5 volume made up of m disks, a request to add n disks and transform the $m+n$ disks into a*

1. ALV is an acronym here for "Aggregate accesses to data chunks, Lazy updates of mapping metadata, and Valve-based rate control."

2. There is a variety of strategies in RAID-5 that evenly distributes the data chunks and parity chunks [9]. The specific data distribution shown here and used throughout this paper is called the left-asymmetric distribution. All conclusions for the other distributions are similar.

new RAID-5 volume is termed a RAID scaling request and is represented by A(m, n).

Definition 2. *Given a RAID scaling request $A(m,n)$ and a chunk x, the state that Chunk x' (for any integer x' and $0 \le x' < x$) has been moved and Chunk x'' (for any integer x'' and $x'' \ge x$) has not been moved is called a RAID scaling state and is represented by $S(m,n,x)$.*

Definition 3. *Given a RAID scaling state S(m, n, x), the reordering window of A(m,n) at chunk x is a window of data chunks that begins with data chunk x and consists of a specified number (window size) of data chunks. Its window size is represented by R(m, n, x), and $R(m,n,x) = Max\{w$: w is a nonnegative integer; for any integers x', $x'' \in [x, x+w)$ and $x' < x''$, Chunks x' and x'' cannot be overwritten by each other or by a parity chunk no matter in what order they are moved$\}$.*

The data redistribution causes an arbitrary data chunk x in the original geometry to be overwritten by either a data chunk x' or a parity chunk in the new geometry. For any state $S(m,n,x)$, two observations are obtained as follows:

Observation 1. If Chunk x is overwritten by Chunk x', Chunks $x, x+1, x+2, \ldots, x'-1$ can be moved in an arbitrary order without overwriting any valid chunk. Chunks $x, x+1, x+2, \ldots, x'-1$ make up a reordering window.

Observation 2. If Chunk x is overwritten by a parity chunk, we suppose that Chunk x'' is the first data chunk greater[3] than the parity chunk in the new geometry. Then Chunks $x, x+1, x+2, \ldots, x''-1$ can be moved in an arbitrary order without overwriting any valid chunk. Chunks $x, x+1, x+2, \ldots, x''-1$ make up a reordering window.

Based on the two observations, we get Theorem 1.

Theorem 1. *The size of a reordering window is*

$$R(m,n,x) = n \times \left\lfloor \frac{x}{m-1} \right\rfloor + \delta,$$

$$\delta = \begin{cases} 0, & r < t \wedge r \le t', \\ -1, & r < t \wedge r > t', \\ 1, & r \ge t \wedge r+1 \le t', \\ 0, & r \ge t \wedge r+1 > t', \end{cases}$$

where $r = x \bmod (m-1)$, $t = m-1-(g \bmod m)$, $t' = m+n-1-(g \bmod (m+n))$, *and* $g = \lfloor x/(m-1) \rfloor$.

Proof. As far as $A(m,n)$ is concerned, in the original geometry, a given chunk x lies in the stripe whose ordinal number is $g = \lfloor x/(m-1) \rfloor$. The parity chunk in Stripe g lies in the disk whose ordinal number is $t = m-1-(g \bmod m)$. Since each stripe consists of $m-1$ data chunks, we denote $r = x \bmod (m-1)$. If $r < t$, Chunk x lies in the disk whose ordinal number is $d = r$; Otherwise, the parity chunk is less[4] than Chunk x, therefore, $d = r+1$.

In the new geometry, the parity chunk in Stripe g lies in the disk whose ordinal number is $t' = m+n-1-(g \bmod (m+n))$. If $d \ne t'$, Chunk x is overwritten by a data chunk x'; otherwise, Chunk x is overwritten by a parity chunk.

First, let us inspect the case where Chunk x is overwritten by Chunk x' (i.e., $d \ne t'$). If $d < t'$, then $x' = g \times (m+n-1) + d$. If $d > t'$, then the parity chunk is less than Chunk x'. Therefore, we get $x' = g \times (m+n-1) + d - 1$. By Observation 1, we can get $R(m,n,x) = x' - x$.

The other case is that Chunk x is overwritten by the parity chunk (i.e., $d = t'$). Suppose that Chunk x'' is the first data chunk greater than the parity chunk in the new geometry. We get $x'' = g \times (m+n-1) + d$. By Observation 2, we have $R(m,n,x) = x'' - x$.

By summarizing the two cases, we get

$$\begin{aligned} R(m,n,x) &= \begin{cases} g \times (m+n-1) + d - x, & d < t', \\ g \times (m+n-1) + d - x, & d = t', \\ g \times (m+n-1) + d - x - 1, & d > t', \end{cases} \\ &= \begin{cases} g \times (m+n-1) + d - x, & d \le t', \\ g \times (m+n-1) + d - x - 1, & d > t'. \end{cases} \end{aligned}$$

Since $g = \lfloor x/(m-1) \rfloor$,

$$d = \begin{cases} x \bmod (m-1), & r < t \\ x \bmod (m-1) + 1, & r \ge t \end{cases},$$

and $\lfloor x/(m-1) \rfloor \times (m-1) + x \bmod (m-1) = x$, we have

$$R(m,n,x) = \begin{cases} n \times \lfloor x/(m-1) \rfloor, & r < t \wedge r \le t', \\ n \times \lfloor x/(m-1) \rfloor - 1, & r < t \wedge r > t', \\ n \times \lfloor x/(m-1) \rfloor + 1, & r \ge t \wedge r+1 \le t', \\ n \times \lfloor x/(m-1) \rfloor, & r \ge t \wedge r+1 > t'. \end{cases}$$

That is,

$$R(m,n,x) = n \times \left\lfloor \frac{x}{m-1} \right\rfloor + \delta,$$

$$\delta = \begin{cases} 0, & r < t \wedge r \le t', \\ -1, & r < t \wedge r > t', \\ 1, & r \ge t \wedge r+1 \le t', \\ 0, & r \ge t \wedge r+1 > t', \end{cases}$$

where $r = x \bmod (m-1)$, $t = m-1-(g \bmod m)$, $t' = m+n-1-(g \bmod (m+n))$, and $g = \lfloor x/(m-1) \rfloor$. □

Corollary 1. *Given m and n, $R(m,n,x)$ increases with x in a step-like manner.*

Proof. In Theorem 1, the first addend, $n \times \lfloor x/(m-1) \rfloor$, increases with x in a step manner, and the value of δ is 0, -1, or 1. Therefore, $R(m,n,x)$ increases with x in a step-like manner. Corollary 1 is verified by the values of $R(3,1,x)$, as shown in Fig. 2. □

Our discovery of a reordering window provides two opportunities for optimizing. First, multiple successive chunks can be accessed via a single I/O. For instance, in a redistribution state during RAID-5 scaling from 3 disks to 4, shown in Fig. 3, the reordering window consists of

3. The order relationship is in the RAID-0 order, same as below.
4. The order relationship is in the RAID-0 order, same as below.

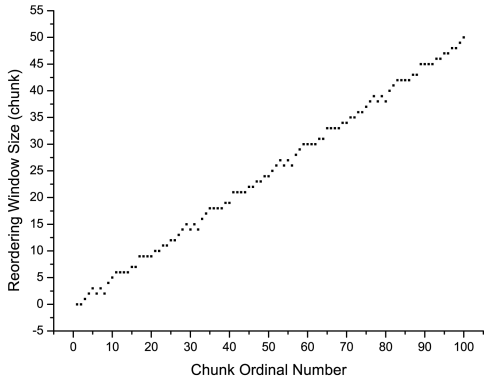

Fig. 2. Values of $R(3,1,x)$ for $x \in [0,100]$. $R(3,1,x)$ increases with x in a step-like manner.

Chunks 18-26. Chunks 18, 20, 24, and 26 can be read via only one I/O, instead of four I/Os; Chunks 19, 22, and 26 can be written via only one I/O, instead of three I/Os (see Fig. 4). Second, after a data chunk in the reordering window is copied, both its new and original replicas are valid. As a result, even if the mapping information is not updated immediately after it is copied, data consistency is still guaranteed. In the above example, we suppose that Chunks 19, 22, and 26 have been copied to their new locations and the mapping metadata have not been updated (see Fig. 4), when the system fails. The original replicas of Chunks 19, 22, and 26 will be used after the system reboots. Chunk copying does not overwrite any valid data, therefore, data remain consistent unless Chunks 19, 22, and 26 have been updated since being copied. That is to say, an unexpected system failure only wastes some data accesses, without compromising data consistency. Therefore, most of metadata updates can be postponed without loss of data consistency.

3 ALV: A Scaling Approach

Leveraging our insight into a reordering window, we propose ALV, an efficient data redistribution approach to RAID-5 scaling. To better understand how ALV works, we will first present an overview of its data redistribution process, followed by a description of the three key technologies in ALV.

3.1 Overview of the Redistribution Process

Before scaling, a mapping equation [10] describes the original geometry, where m disks serve user requests. Fig. 5 illustrates an overview of the redistribution process that is caused by $A(m,n)$. ALV uses *a sliding window* to describe the mapping information of a continuous segment in a RAID-5 volume under scaling. During scaling, only the data that lie within the sliding window are copied to new locations. It should be noted that a sliding window is

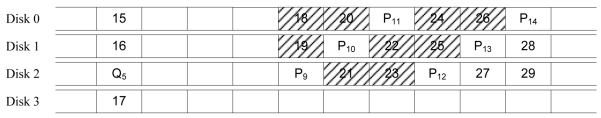

Fig. 3. A state of data redistribution (disk 3 is newly added).

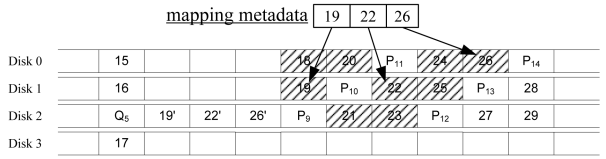

Fig. 4. If data chunks are copied to their new locations and metadata are not yet updated when the system fails, data consistency is still maintained because the data in their original locations are valid and available.

different from a reordering window. The former is a mapping-management solution during RAID scaling, while the latter is a characteristic in the process of RAID scaling. To meet the restriction of data consistency, a sliding window should be no larger than the corresponding reordering window.

An incoming user request is mapped in one of the following three ways according to its logical address:

- If it is greater than any data chunk in the sliding window, it is mapped through the original geometry, where m disks serve user requests.
- If it is less than any data chunk in the sliding window, it is mapped through the new geometry, where $m+n$ disks serve user requests.
- If its logical address is within the range of the sliding window, it is mapped through the sliding window (see Fig. 8 for more details).

When all of the data in a sliding window are moved, the sliding window slides ahead by one window size. In this way, the newly added disks are gradually available to serve user requests. After the sliding window reaches the end of the original volume, no data are read for redistribution. When data redistribution is completed, a new mapping equation is used to describe the new geometry.

3.2 Aggregate Accesses to Data Chunks

The reordering window characteristic enables data chunks within a sliding window to be moved in an arbitrary order. Since the transfer rate of a disk drive increases with the size of a disk access, ALV changes the movement order of data chunks to access multiple successive chunks via a single I/O. Take the state $S(3,1,18)$ as an example for which we have $R(3,1,18)=9$. As shown in Fig. 6, ALV issues the first I/O request to read Chunks 18, 20, 24, and 26; the second request to read Chunks 19, 22, and 25; and the third request for Chunks 21 and 23 simultaneously. By this means, to read all of these chunks, ALV requires only three I/Os, instead of nine.

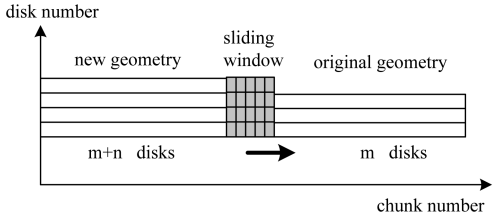

Fig. 5. An overview of the data redistribution process. As the sliding window slides ahead, the newly added disks are gradually available to serve user requests.

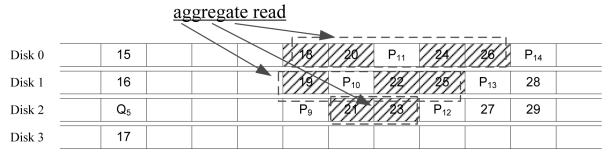

Fig. 6. Aggregate reads for $S(3,1,18)$. Multiple successive chunks are read via a single I/O.

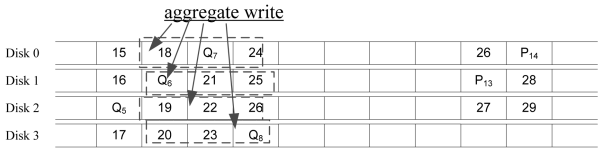

Fig. 7. Aggregate writes for $S(3,1,18)$. Multiple successive chunks are written via a single I/O.

When all nine chunks are read into a memory buffer and all new parity chunks are computed, ALV issues the first I/O request to write Chunk 18, a new parity chunk Q_7, and Chunk 24; the second I/O to write a parity chunk Q_6 as well as Chunks 21 and 25; the third I/O for Chunks 19, 22, and 26; and the fourth I/O for Chunks 20, 23, and a parity chunk Q_8 simultaneously (see Fig. 7). In this way, only four write requests are issued as opposed to 12.

In the previous case, nine data chunks reside in the memory at some time, so the memory space needs to be reserved to hold the nine chunks. The set of data chunks that is read into the memory via a group of asynchronous I/Os is called *an aggregation window*. For $A(m,n)$, m reads and $m+n$ writes are required to redistribute all of the data in an aggregation window. Clearly, the size of an aggregation window is not fixed but tunable. Provided that the size of the aggregation window is six, the first six chunks (i.e., Chunks 18-23) are first read into the memory. In this case, the first read request reads Chunks 18 and 20; the first write request writes Chunk 18 and a parity chunk Q_7. To move all these nine chunks, five reads and eight writes are required.

The access aggregation converts sequences of small requests into fewer but larger requests. The cost of one seek is prorated over multiple chunks. Moreover, because full-stripe writes are more efficient than partial-stripe writes, the optimal chunk size for RAID-5 is relatively small [11]. A typical choice is 32 or 64 KB [3], [12], [13], [10]. Thus, accessing multiple successive chunks via a single I/O enables ALV to have a larger redistribution throughput. Since data densities in disks increase at a much faster rate than improvements in seek times and rotational speeds, access aggregation grows profitable.

When writes for redistributing data are not aligned among all of the disks, RAID-5 array controller first reads the old user's data and the old corresponding parity information. Then, it calculates the new parity information using the old data, the new data, and the old parity information [14], [7]. ALV uses *the write alignment technique* to eliminate extra reads to old user's data and old parity (see Fig. 7). The size of an aggregation window is exactly a multiple of a stripe in the new geometry. While using ALV, RAID-5 array controller calculates the new parity information using only the new data. The write alignment technique can improve the performance of data redistribution.

3.3 Lazy Updates of Mapping Metadata

While the data redistribution is in progress, the RAID storage serves user requests. Furthermore, the coming user I/Os may be write requests whose target addresses are within the current sliding window. As a result, if ALV writes mapping metadata simply after all of the chunks within the sliding window are moved, data consistency may be destroyed. ALV uses lazy updates of mapping metadata to minimize the number of metadata writes without loss of data consistency.

The key idea behind lazy updates is that the mapping metadata are not updated onto the disks (a.k.a., *checkpoint*) until a threat to data consistency appears. Fig. 8 shows a state transition diagram describing lazy updates of mapping metadata, where moving all of the data within an aggregation window is an operation unit.

- Once all of the data within an aggregation window have been moved, the aggregation window goes ahead by one aggregation window size (State 0).
- The mapping metadata are not updated until one of the two states appears: 1) when the aggregation window reaches the end of the current sliding window (State 1), the data redistribution is checkpointed and a new sliding window is initialized (State 2); 2) when a user write request arrives in the region where data have been moved and the movement has not been checkpointed (State 3), the data redistribution is checkpointed and a new sliding window is initialized (State 4) before the write request is served.
- In States 2 or 4, once all of the data in the first aggregation window have been moved, the redistribution state transmits to State 0.

Lazy updates of mapping metadata decrease the number of metadata writes significantly, since one metadata write stores the map changes of many data chunks. In the best case, moving all of the chunks within a sliding window requires only one metadata write. Furthermore, the reordering window characteristic enables lazy updates to guarantee data consistency. Even if the system fails unexpectedly when some chunks have been copied and their mapping metadata have not been updated, only some of the data accesses are wasted. The data consistency is not destroyed since the chunks in their original locations are valid. It should be noted that the probability of a system failure is very low.

3.4 Valve-Based Rate Control

During scaling a RAID-5 volume, data redistribution and foreground applications share and even contend for the I/O resources in the system. ALV keeps track of the application workload on the RAID-5 volume, and further, adjusts the redistribution speed dynamically.

ALV uses an on/off logical valve to adjust the redistribution rate. Whether data redistribution performs in the next period of time P_{i+1} depends on application workload W_i in the current period P_i. As shown in Fig. 9, in the period P_{i+1}, data redistribution performs if W_i is relatively low, whereas it is throttled (maybe by thread sleeping) if W_i is

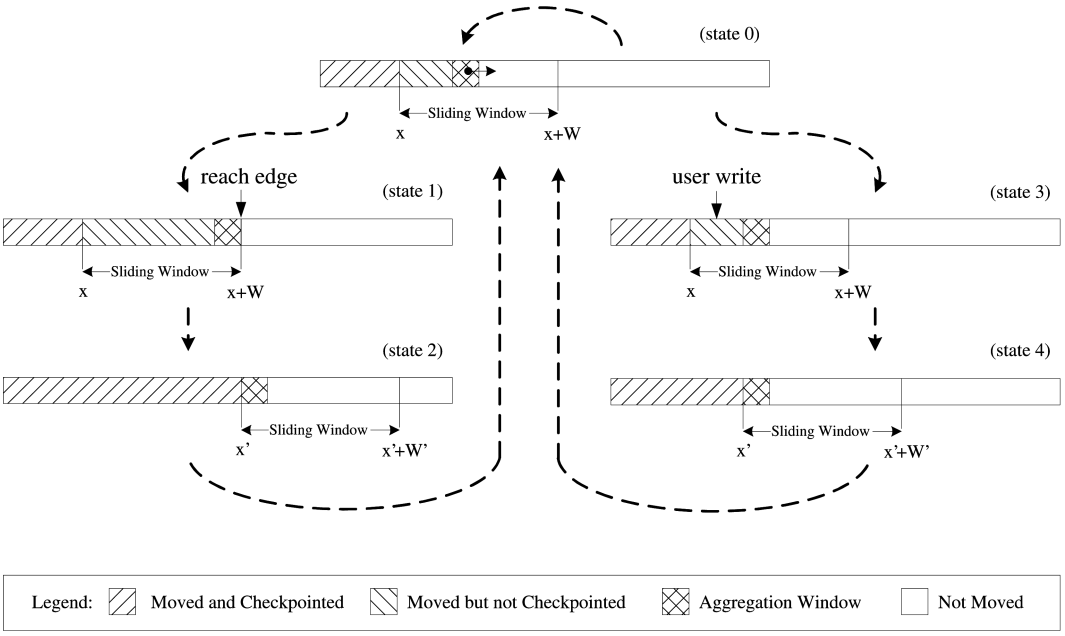

Fig. 8. The state transition diagram describing lazy updates of mapping metadata. Data redistribution is checkpointed only when the aggregation window reaches the edge of the current sliding window (state 1) or when a user write request arrives in the region, where data have been moved but the movement has not been checkpointed (state 3).

high. This means that we use W_i to approximate W_{i+1}. Therefore, the period length should not be too large.

Since storage QoS is measured in throughput and latency, ALV monitors two performance metrics inside the volume: system congestion and service time. When system congestion appears, at least one user I/O cannot enter the volume for service immediately. The dominating reason for congestion is that the volume buffers are exhausted temporarily. ALV also monitors the latency[5] for every user I/O. Let N_i be the number of user I/Os completed in P_i. Let $L_i(k)$ be the latency of the kth user I/O in P_i. Let the latency bound for the period P_i be denoted by B_i. We define the latecomer ratio LR_i as the fraction of user I/Os whose latency is greater than B_i: $LR_i = |\{k : L_i(k) > B_i, k = 1, \ldots, N_i\}|/N_i$. When system congestion occurs in the period P_i, or when $LR_i > \theta$, the workload W_i is adjudicated to be high, and vice versa. Here, θ is called *"a slack factor"*. If $\theta = 0$, it will cause unaffordable overprovisioning and unsatisfactory redistribution speeds due to highly variable service times in storage systems [15] and workload fluctuations on arbitrary time scales [5].

Existing adaptive schemes for rate control regard a storage system as a black box and detect the application workloads outside the storage system by measuring the performance perceived by the applications [16], [15]. Consequently, changes in the load that the storage system endures actually cannot be detected very quickly, due to the prefetching and caching in the file systems. Conversely, ALV monitors the load inside the volume, which makes it possible to quickly obtain very accurate information about the load. This makes the control actions more responsive. In addition, ALV provides a control parameter B_i for an application-level rate control.

5. It is not the latency perceived by applications, but rather the time that RAID-5 controller processes an I/O request.

4 Implementation

We implemented ALV in the MD driver shipped with Linux Kernel 2.6.18. MD is a software RAID system, which uses MD-Reshape to scale RAID-5 volumes [3]. Implementing ALV in MD makes it convenient to make a performance comparison between ALV and MD-Reshape. About 500 lines of code, counted by the number of semicolons and braces, were modified or added to the MD driver.

When RAID scaling $A(m, n)$ begins, MD creates the data mover, a kernel thread to perform the data redistribution. ALV uses three variants to track how the expansion is progressing and determine the movement stage that the target part of the volume involves (see Fig. 8). ALV cannot redistribute a new aggregation window until all of the I/O requests, already issued to this window, are completed. To redistribute an aggregation window, ALV performs three steps in turn:

- ALV simultaneously sends m I/Os to read all of data chunks within the aggregation window.
- Once all of these I/Os are completed, ALV fiddles with pointers to shuffle the data chunks, and it calculates the parity.

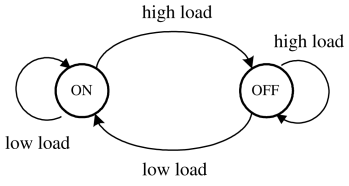

Fig. 9. The state transition diagram describing valve-based rate control. Data redistribution performs when the valve is on, whereas data redistribution is throttled when the valve is off.

- ALV simultaneously sends $m + n$ I/Os to write all of data chunks within the aggregation window and the corresponding parity data.

While an aggregation window is being redistributed, any I/O attempt into the aggregation window is blocked until the redistribution of this window is finished. ALV wakes up all of the I/Os queuing up at the aggregation window once this redistribution is over.

When a user I/O arrives, ALV detects whether the system is congested. When a user I/O is completed, ALV also verifies whether this I/O is a latecomer. If ALV concludes that the workload pressure is high in the current period of time, the data mover thread sleeps for a period of time; otherwise, the thread performs data redistribution continuously for a period. In our experiments, a period was set one second.

5 Performance Evaluation

This section presents results of a comprehensive experimental evaluation comparing ALV with MD-Reshape. This performance study analyzes their performance in terms of user response time and redistribution time.

5.1 Evaluation Methodology

We evaluated our design by running trace-driven experiments over a real system. To replay I/O traces, we implemented a block-level replay tool using POSIX asynchronous I/O. It opens a block device with the O_DIRECT option, and issues an I/O request, when appropriate, according to trace files. When an I/O request is completed, it gathers the corresponding response time. Each experiment lasted from the beginning to the end of the data redistribution.

Our experiments used the following three real-system disk I/O traces with different characteristics:

- TPC-C traced disk accesses of the TPC-C database benchmark with 20 warehouses [17]. It was collected in 2001 with one client running 20 iterations.
- $Cello$-99 was collected on an HP UNIX server with 2 GB memory in 1999 [18]. The I/O accesses were filtered by the file system cache. Therefore, its temporal locality is quite poor. The Cello-99 workload is also highly bursty.
- SPC-Web is from Storage Performance Council (SPC) [19], a vendor-neutral standards body. It was collected from a system running a Web search engine. This trace is read dominant and with high locality [20].

The testbed used in these experiments is described as follows: Linux kernel 2.6.18 was installed on a 2.4 GHz Intel Xeon machine with 1 GB of memory and an Emulex LP982 HBA card. The file system used was EXT3. Via a Brocade Silkworm 3800 FC switch, this machine was connected with an FC JBOD controlling several Seagate ST3146807FC disks.

A group of rate-control parameters means a trade-off between the redistribution time objective and the response time objective. Furthermore, unless both redistribution time and user response time, using the one approach, are, respectively, smaller than those using the other approach, we do not know if we can predict that the former approach outperforms the latter. Therefore, for convenient comparison, we chose control parameters for the different experiments.

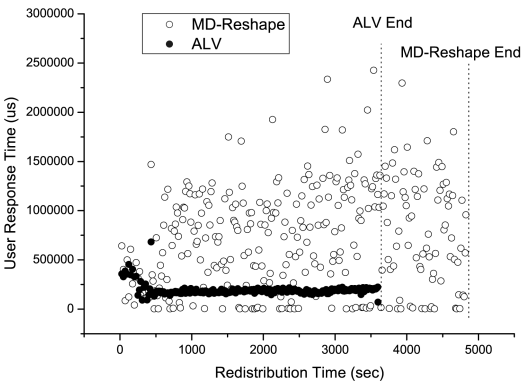

Fig. 10. Performance comparison between MD-Reshape and ALV under the TPC-C workload.

For comparing user response time, we collected the response time of each user request, broke the response-time sequence into equal-sized short sequences, and obtained an average response time from each short sequence.

5.2 Performance Advantages of ALV over MD-Reshape

The purpose of our first experiment was to quantitatively characterize the advantages of ALV through a comparison with MD-Reshape. We conducted a RAID-scaling request $A(3, 1)$, where each disk had a capacity of 30 GB. Each approach performed with the 64 KB chunk size under a TPC-C workload. In this experiment, ALV reserved the 2 MB memory space for each member disk to hold an aggregation window. To provide a fair comparison, we increased the stripe cache size to 512 4 KB-buffers per disk for MD-Reshape. "Sync_speed_max" and "sync_speed_min" in MD-Reshape were set 200 and 1 MB/s, respectively. The latency bound in ALV was set 15 ms statically and the slack factor was set 0.1. This parameter setup acts as the baseline for the latter experiments from which any change will be stated explicitly.

Fig. 10 illustrates that, as a result of effective exploitation of the reordering window characteristic, ALV demonstrated a noticeable improvement over MD-Reshape in two metrics. First, the redistribution time using ALV was obviously shorter than that using MD-Reshape. They were 3,600 and 4,828 seconds, respectively. In other words, ALV had a 25.43 percent shorter redistribution time than MD-Reshape. One of the main factors in ALVs reducing the redistribution time was the decline of the redistribution I/Os via access aggregation. Another main factor was the decline of metadata writes by lazy updates. As shown in Table 1, the number of the redistribution I/Os issued by ALV was only 0.29 percent of that by MD-Reshape; the number of the metadata updates performed by ALV was only 0.28 percent of that by MD-Reshape.

Second, ALV more significantly outperformed MD-Reshape in the user response time. A study of the distribution of user response times during the data redistribution (Fig. 11) showed that ALV had a markedly smaller impact on the user response times than MD-Reshape did. The average user response time in the

TABLE 1
The Number of Data Accesses and Metadata Updates with the Two Approaches

	MD-Reshape	ALV	ratio
data reads	15728640	46080	0.29%
data writes	31457280	92160	0.29%
md updates	9988	28	0.28%

redistribution using MD-Reshape was 738.147 ms, whereas that using ALV was 192.608 ms. That is to say, ALV reduced the average I/O latency experienced by applications by as much as 73.91 percent with respect to MD-Reshape. This improvement occurred because of two reasons: 1) the decline in both data accesses and metadata writes decreased the disk queue lengths. Moreover, this significantly alleviated the redistribution's interference with the application I/Os, and thereby, enabled the applications to obtain more sequential accesses. The throughput of a sequential pattern can be an order of magnitude higher than that of a random pattern [21]. 2) The adaptive rate control in ALV alleviated the adverse impact of the data redistribution on the application performance.

Fig. 10 also demonstrates the difference in the quality of rate control using MD-Reshape and ALV. MD-Reshape caused excessive oscillation in the user response time throughout the redistribution. In comparison, ALV rapidly reduced the user response time to a lower level, and preserved this level steadily until the redistribution ended. The reason MD-Reshape caused excessive oscillation was that its funnel-like scheme for rate control could not adapt to great changes of application behaviors over time. Differently, ALV achieved a satisfactory stability because it could detect the accurate workload pressure quickly, and in turn, it provided a real-time control to the redistribution rate using an on/off logical valve.

5.3 Impact of the Application Workload

A factor that might affect the benefits of ALV is the workload under which data redistribution performs. We measured the performance of MD-Reshape and ALV to perform $A(3,1)$ under the Cello-99 and SPC-Web workloads.

For the Cello-99 workload, "sync_speed_max" and "sync_speed_min" in MD-Reshape were set 10 and 1 MB/s.

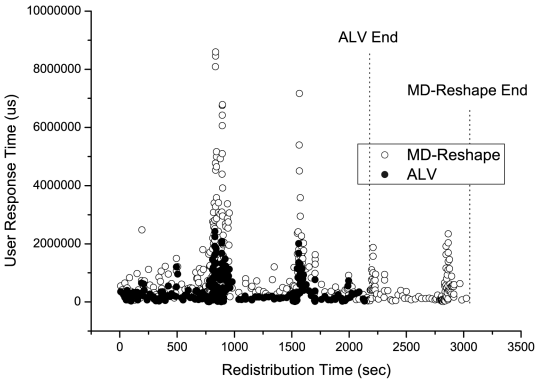

Fig. 12. Performance comparison between MD-Reshape and ALV under the Cello-99 workload.

Fig. 12 shows the measured redistribution times and user response times using MD-Reshape and ALV. It again revealed the efficacy of ALV in improving the redistribution time and user response time simultaneously. The redistribution times using MD-Reshape and ALV were 3,027 and 2,141 seconds, respectively. That is to say, ALV brought an improvement of 29.27 percent in the redistribution time. The average user response times using MD-Reshape and ALV were 1,108.110 and 444.381 ms. In other words, ALV enhanced the user response time by 59.90 percent.

For the SPC-Web workload, "sync_speed_max" and "sync_speed_min" in MD-Reshape were set 7 and 1 MB/s, respectively. Fig. 13 plots the user response time using MD-Reshape and ALV as the data redistribution proceeded. Similarly, ALV brought an improvement in the redistribution time and user response time simultaneously. The redistribution times using MD-Reshape and ALV were 4,338 and 3,294 seconds. That is to say, ALV had a 24.07 percent shorter redistribution time than MD-Reshape. The average user response times were 484.130 and 226.023 ms, respectively. In other words, ALV improved the user response time by 53.31 percent.

To compare the performance of ALV under different workloads, Fig. 14 shows the improvement in the redistribution time and user response time by ALV, as compared with MD-Reshape. For completeness, we also conducted a comparison experiment on the redistribution time with no loaded workload. To scale a RAID-5 volume

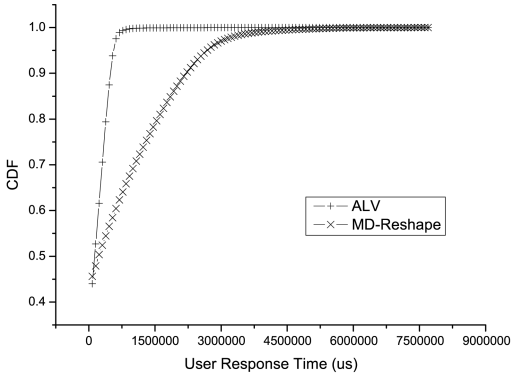

Fig. 11. Cumulative distribution of user response times during the data redistributions by the two approaches under the TPC-C workload.

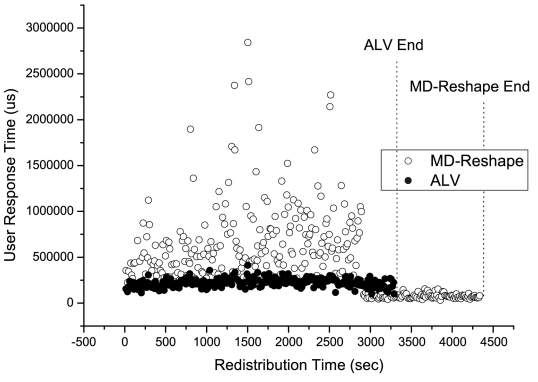

Fig. 13. Performance comparison between MD-Reshape and ALV under the SPC-Web workload.

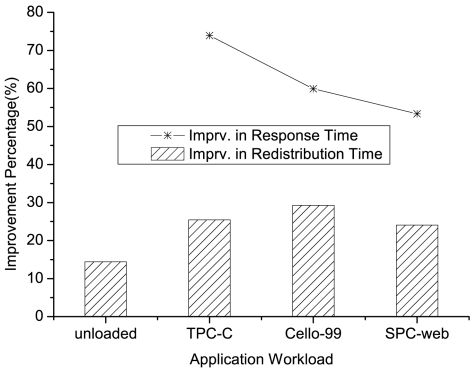

Fig. 14. Comparison of performance improvement by ALV over MD-Reshape under different workloads. The label "unloaded" means the improvement when scaling a RAID-5 volume offline.

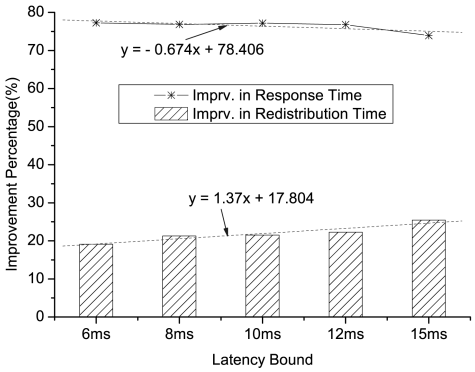

Fig. 15. Comparison of performance improvement by ALV over MD-Reshape with different latency bounds. In the functions, x does not stand for a latency bound accurately but an ordinal number in the x-axis, i.e., not 6, 8, 10, 12, and 15 ms, but 1, 2, 3, 4, and 5. This is very enough for a qualitative analysis on how two aspects of the improvement vary with regard to the change in the latency bound.

offline, MD-Reshape used 1,836 seconds whereas ALV consumed only 1,571 seconds. ALV provided an improvement of 14.43 percent in the redistribution time.

We drew two conclusions from Fig. 14. First, the improvement in redistribution time under any workload was noticeably higher than that with no loaded workload. There were two reasons for this phenomenon: 1) the application accesses resulted in a disturbance to the sequence of disk seeks required by the data redistribution; 2) MD-Reshape issued small redistribution requests. As a result, the application accesses affected the redistribution efficiency using MD-Reshape more adversely due to the frequent long seeks.

Second, under various workloads, ALV consistently outperformed MD-Reshape by 53.31-73.91 percent in the user response time and by 24.07-29.27 percent in the redistribution time simultaneously. Compared with the improvement under the TPC-C workload, the improvement under the Cello-99 workload, in the redistribution time, was higher, while that in the user response time was lower. This was due to another different choice in the rate-control spectrum, which led to new allotment of the benefit by ALV between the data redistribution and the application accesses. Most interesting was that the two aspects of improvement under the SPC-Web workload were simultaneously lower than those under the TPC-C workload. This was because the SPC-Web trace is absolutely read dominant (99.98 percent) [20], and in turn, the workload suffered from nearly no extra writes of parity. As a result, the application accesses disturbed the sequence of disk seeks required by the data redistribution in a low level. Hence, the improvement by ALV under the SPC-Web workload was slightly smaller.

5.4 Sensitivity Analysis on the Rate-Control Parameters

Our valve-based rate control is dependent on two parameters: the latency bound and the slack factor. In order to examine the effect of our rate control more clearly, we first conducted some experiments using ALV under the TPC-C workload, with latency bounds of 6, 8, 10, and 12 ms.

We compared the performance of ALV in these experiments with that of MD-Reshape in the baseline experiment shown in Section 6.2. Due to the space limit, we did not present the measured redistribution time and user response time for each latency bound. Instead, we only plotted the improvement by ALV over MD-Reshape in Fig. 15. This comparison indicated that with various latency bounds, ALV consistently outperformed MD-Reshape by 73.91-77.27 percent in the user response time and by 19.08-25.43 percent in the redistribution time simultaneously. This means that ALV offers the administrator a long spectrum of rate-control choices that provide a higher application performance and a shorter redistribution time.

We conducted a qualitative analysis on how two aspects of the improvement varied with regard to the change in the latency bound. Using linear regression, we estimated that the slope coefficient for the improvement in redistribution time was 1.37 (see Fig. 15). This positive coefficient indicated that the improvement by ALV in the redistribution time increased with the latency bound. Because the experiments using ALV were compared with the same baseline experiment using MD-Reshape, this also means that ALV performed the data redistribution faster with a larger latency bound. However, this speedup of data redistribution came at a cost of the application performance. The negative slope coefficient (i.e., -0.674) for the improvement in the user response time indicated that the improvement by ALV in the user response time decreased with the latency bound. In other words, users in the ALV redistribution with a larger latency bound experienced a bit higher I/O response time. The reason was that more disk bandwidth was dedicated to the data redistribution.

Second, we performed a sensitivity analysis on the slack factor to show how various values affect the degree of interference between the data redistribution and the foreground applications. We set the latency bound to 15 ms and changed the slack factor from 0.0 to 0.2. As shown in Fig. 16, as the slack factor increased, the improvement in the redistribution time increased, while that in the user response time decreased. Two extreme cases proved to be the most interesting. When the slack factor was 0.0, the user response time was improved significantly, while the redistribution time was prolonged, as compared with MD-Reshape. Conversely, when the slack factor was 0.2, the redistribution time was reduced more significantly,

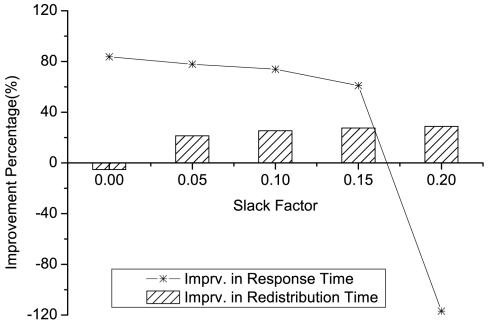

Fig. 16. Comparison of performance improvement by ALV over MD-Reshape with different slack factors.

while the user response time was worsened, as compared with MD-Reshape.

5.5 Impact of the Chunk Size

To examine the impact caused by the different chunk sizes, we conducted a performance evaluation on MD-Reshape and ALV. They were to perform $A(3,1)$ with chunk sizes of 16, 32, 128, and 256 KB under the TPC-C workload. "Sync_speed_max"(s) in MD-Reshape were set 8, 10, 10, and 8 MB/s, respectively; "sync_speed_min"(s) were set 1, 3.5, 1, and 1 MB/s, respectively.

Fig. 17 plots the improvement by ALV over MD-Reshape in the redistribution time and the average response time. One can see that, for all of the examined chunk sizes, ALV consistently outperformed MD-Reshape in both the redistribution time and user response time. The improvement in the redistribution time fluctuated between 25.43 and 34.31 percent. Simultaneously, the improvement in the user response time stabilized between 73.91 and 77.64 percent.

The result of the linear regression indicated that two aspects of the improvement were relatively steady with relation to the chunk size. The reason for a slight increase (due to their positive slope coefficient) was that the metadata updates in the ALV redistribution became fewer with a larger chunk size. MD-Reshape writes mapping metadata every 3 MB of data, regardless of the chunk size. Therefore, MD-Reshape wrote mapping metadata for the same times with different chunk sizes. However, since the size of a reordering window is independent of the chunk size, to scale a given volume under the same workload, ALV wrote mapping metadata for a fewer times with a larger chunk size.

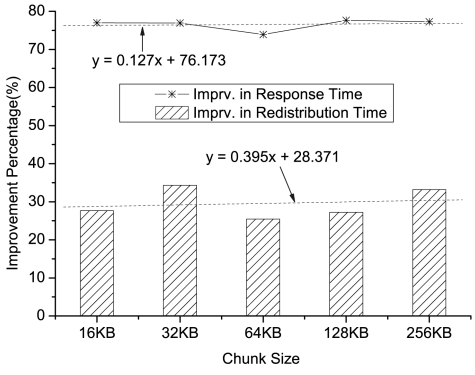

Fig. 17. Comparison of performance improvement by ALV over MD-Reshape with different chunk sizes. In the functions, x does not stand for a chunk size accurately but an ordinal number in the x-axis, i.e., not 16, 32, 64, 128, and 256 KB, but 1, 2, 3, 4, and 5. This is enough for a qualitative analysis on how two aspects of the improvement vary with regard to the change in the chunk size.

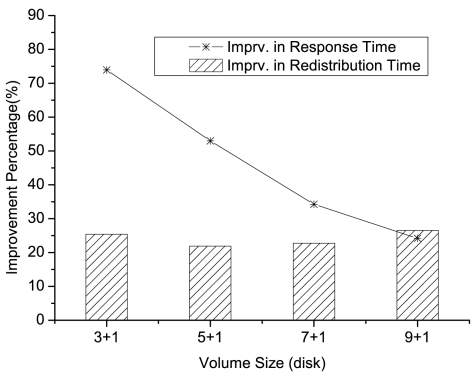

Fig. 18. Comparison of performance improvement by ALV over MD-Reshape with different volume sizes.

5.6 Impact of the Volume Size

We studied the effect of the volume size by varying the volume size. We measured the performance of MD-Reshape and ALV to perform RAID scaling requests of $A(5,1)$, $A(7,1)$, and $A(9,1)$ under the TPC-C workload. "Sync_speed_max"(s) in MD-Reshape, with volume sizes of 5, 7, and 9 disks, were set 6, 4.5, and 3.5 MB/s, respectively.

Fig. 18 shows a comparison of the performance improvement by ALV over MD-Reshape as the volume size increases along the x-axis. It can be seen that the improvement in the redistribution time stabilized between 21.94 and 26.55 percent. Simultaneously, as the number of disks in the RAID-5 volume increased, the improvement in the user response time decreased. The reason for the decreased improvement at a higher volume size was that the same trace was applied to all of the volume sizes. As a result, with more disks, the lower I/O request intensity was imposed on an individual disk, which implied that a less interference would be avoided by ALV. However, as the overall I/O traffic intensity increases proportionally to the volume size, we expect the improvement by ALV to be more significant. Moreover, since more disks in a RAID-5 volume lead to a lower mean time between failures (MTBFs) for the volume, it is a common practice to limit the number of disks in a RAID-5 volume.

5.7 Impact of the Buffer Size

Finally, to examine the impact of the buffer size on the performance of MD-Reshape and ALV, we conducted several experiments with buffer sizes of 1, 2, 3, and 4 MB. For completeness, the experiments included two parts: one with no loaded workload and another under the TPC-C workload.

When no workload was loaded, ALV behaved well with all of the buffer sizes. With the default rate-control parameter setup, however, the behavior of MD-Reshape was quite unexpected. As the buffer size increased from 1 to 2 MB, the redistribution time decreased from 2,345 to 1,836 seconds. However, the redistribution time increased to 8,255 seconds when the buffer size was 3 MB. This strange phenomenon was because the mechanism for rate

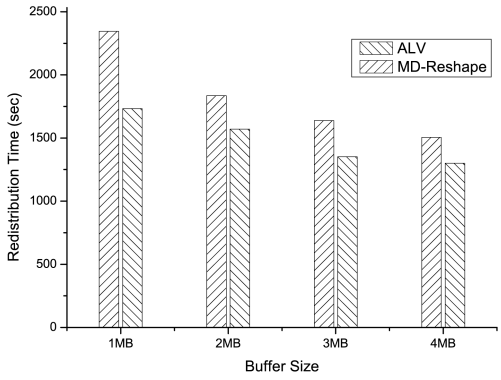

Fig. 19. Performance comparison between MD-Reshape and ALV with different buffer sizes when no workload is loaded.

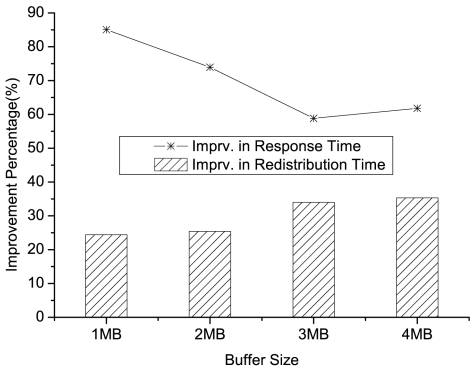

Fig. 20. Comparison of performance improvement under the TPC-C workload with different buffer sizes.

control slowed down the redistribution deliberately. To avoid the rate control from impairing the redistribution efficiency, we set "sync_speed_min" to a large value (i.e., 150 MB/s). As shown in Fig. 19, the redistribution time using MD-Reshape and ALV decreased as the buffer size increased along the x-axis. With each buffer size, ALV outperformed MD-Reshape consistently. The improvement by ALV fluctuated between 13.68 and 26.10 percent.

Under the TPC-C workload, "sync_speed_min"(s) in MD-Reshape with buffer sizes of 1, 3, and 4 MB were set 1, 3.5, and 4 MB/s, respectively, while "sync_speed_max"(s) were all set 10 MB/s. Fig. 20 plots the improvement in redistribution time and user response time by ALV over MD-Reshape. It indicates that, with different buffer sizes, ALV consistently outperformed MD-Reshape by 58.83-85.05 percent in the user response time and by 24.39-35.33 percent in the redistribution time simultaneously.

5.8 Result Summary

The experimental results are summarized as follows:

1. With no loaded workload, ALV shortened the redistribution time by 14.43 percent, as compared with MD-Reshape.
2. Under various workloads, ALV consistently outperformed MD-Reshape by 53.31-73.91 percent in the user response time and by 24.07-29.27 percent in the redistribution time simultaneously.
3. As the latency bound or the slack factor in ALV increased, the improvement by ALV in the redistribution time increased, whereas that in the user response time decreased.
4. With different chunk sizes, the improvement by ALV over MD-Reshape was relatively steady.
5. As the number of disks in a RAID-5 volume increased, the improvement by ALV decreased slightly.
6. With different buffer sizes, whether with or without loaded workload, ALV outperformed MD-Reshape noticeably.

6 Related Work

6.1 Disk Addition into RAID-5 Volumes

The HP AutoRAID [13] allows an online capacity expansion. But the system cannot add new disks into an existing RAID-5 volume. Newly created RAID-5 volumes use all of the disks in the system, but previously created RAID-5 volumes continue to use only the original disks.

A patent [22] presents a method to eliminate the need to rewrite the original data blocks and parity blocks on original disks. However, the method makes parity blocks be either only on original disks or only on new disks. The obvious distribution nonuniformness of parity blocks will bring into a performance penalty.

Franklin and Wong [23] presented a RAID scaling method using spare space with immediate access to new space. First, the old data are distributed among the set of data disk drives and at least one new disk drive while, at the same time, new data are mapped to the spare space. Upon completion of the distribution, the new data are copied from the spare space to the set of data disk drives. This method requires spare disks available in the RAID.

In another patent, Hetzler [24] presented a method to RAID-5 scaling, namely MDM. MDM exchanges some data blocks between original disks and new disks. MDM can perform RAID scaling with reduced data movement. However, it keeps the data storage efficiency unchanged after scaling. When ALV is exploited, the data storage efficiency is maximized, which many practitioners consider desirable.

D-GRAID [25] restores only live file system data to a hot spare so as to recover from failures quickly. Likewise, it can accelerate the redistribution process if only the live data blocks from the perspective of file systems are redistributed. However, this needs for semantically smart storage systems. Differently, ALV is independent of file systems and can work with any ordinary disk storage.

Gonzalez and Cortes [2] proposed a GA algorithm to control the overhead of scaling a RAID-5 volume. However, GA has a large redistribution cost for two reasons. First, it redistributes one stripe in the new geometry repeatedly. Therefore, it reads or writes only one chunk via an I/O. Second, to enable the redistribution process to restart at the same point when the system comes up from a failure, GA writes mapping metadata onto disks immediately after redistributing each stripe.

The reshape toolkit in the Linux MD driver (MD-Reshape) [3] writes mapping metadata for each fixed-sized data window, instead of doing so for only one stripe. However, all of the user requests to the data window have

to queue up and wait for processing until all of the data chunks within the window are redistributed. Therefore, MD-Reshape cannot reduce the number of metadata updates by simply enlarging the data window. On the other hand, MD-Reshape issues very small (4 KB) I/O operations for data redistribution. This limits the redistribution performance due to there being more disk seeks.

6.2 Rate Control in Data Migration

The most intuitive control scheme is to issue migration requests in a low priority in order to minimize adverse impact of data migration on application performance [13], [26], [20]. Under complicated workloads, however, how to assign the relative priorities is still an open problem. The second scheme, as used in MD-Reshape [3], uses an upper bound and/or a lower bound to control the migration rate, regardless of application states. Because application behaviors vary greatly over time, this funnel-like scheme is unlikely to behave well. With a redistribution deadline guaranteed, the third scheme, like GA [2], tries to use idle periods of disks to perform data redistribution. This scheme controls the degradation of application performance by specifying different redistribution deadlines. With unpredictable workloads, however, it is difficult to predefine a reasonable deadline.

A feedback-based method, called MS Manners [27], slows down the run rate of the low-importance process when its progress slows. It cannot provide guarantees to important tasks because it only monitors the progress of low-importance processes. Lu et al. [16] proposed a control-theoretic approach to control the rate of data migration. However, it is a challenge to find a function that describes the sensitivity of average user latency with regard to the change in migration rate, especially when application workloads are not steady. Differently, ALV uses a simple on/off logical valve to adjust the redistribution rate. Hence, it is unnecessary to discover the relation between user latency and migration rate.

6.3 Failure Recovery in RAID-5

When a member disk in a RAID-5 volume fails, all of the lost data must be rebuilt and written to a spare disk quickly [28]. Holland et al. [26] found that disk-oriented rebuilding outperforms stripe-oriented rebuilding. The former reads multiple stripes from all surviving disks simultaneously, while the latter proceeds ahead by one stripe. Through simulation experiments, Fu et al. [29] demonstrated that a larger size of a rebuild unit[6] results in a shorter rebuild time and a higher user response time. Hou et al. [30] found that acceptable I/O response and rebuild times can be obtained by choosing a single track as the rebuild unit. Unlike all of the previous methods that perform a sequential rebuild, PRO [20] integrates workload characteristics into the reconstruction algorithm, and rebuilds frequently-accessed areas prior to rebuilding infrequently accessed areas.

Our basic ideas for enhancing the efficiency of RAID-5 scaling were inspired by these results on failure recovery. However, there is an obvious difference between failure recovery and disk addition: no data are overwritten when the former writes the data on the spare disk in an arbitrary order, while some data chunks may be overwritten if the latter redistributes data chunks in an arbitrary order. Fortunately, our discovery of a reordering window in data redistribution alleviates this difference.

6. A rebuild unit is the amount of rebuild data, which is atomically read from each surviving disk.

7 Conclusions and Future Work

This paper makes the following contributions:

- An insightful opportunity for performance optimization is presented: when scaling a RAID-5 volume, there is always a reordering window within which data consistency can be maintained while changing the order of data movements.
- ALV, an efficient data redistribution approach to RAID-5 scaling is proposed. First, ALV changes the order of data movements to access multiple successive chunks via a single I/O. Second, ALV updates mapping metadata lazily to minimize the number of metadata writes. Third, ALV uses valve-based rate control to adjust the redistribution rate adaptively.
- ALV is implemented in Linux Kernel 2.6.18, and its performance is evaluated through a comparison with the conventional approach MD-Reshape. The results demonstrated that ALV consistently outperformed MD-Reshape by 53.31-73.91 percent in the user response time and by 24.07-29.27 percent in the redistribution time under the TPC-C, Cello-99, and SPC-Web workloads.

Similar to scaling up (i.e., adding disks into a RAID-5 volume), scaling down (i.e., disk removal) also needs redistributing data. The only difference is that scaling down performs data redistribution from the end to the beginning of the volume, while scaling up does it in the opposite order. There is also a reordering window during redistribution for scaling down. In future, we will extend the ALV approach for RAID-5 scaling down.

Acknowledgments

The authors would like to thank the anonymous reviewers for their suggestions on improving this paper. This work was supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2007CB311100, the National High Technology Research and Development Program of China under Grant No. 2009AA01Z139, the National Natural Science Foundation of China under Grants 60903183 and 60873066, and the Specialized Research Fund for the Doctoral Program of Higher Education of China under Grant No. 20070003092.

References

[1] D. Patterson, "A Simple Way to Estimate the Cost of Downtime," *Proc. 16th Large Installation Systems Administration Conf. (LISA '02)*, pp. 185-188, Oct. 2002.

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

[3] N. Brown, "Online RAID-5 Resizing. Drivers/md/Raid5.c in the Source Code of Linux Kernel 2.6.18," http://www.kernel.org/, Sept. 2006.

[4] M. Wachs, M. Abd-El-Malek, E. Thereska, and G. Ganger, "Argon: Performance Insulation for Shared Storage Servers," *Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07),* Feb. 2007.
[5] S. Gribble, G. Manku, E. Roselli, and E. Brewer, "Self-Similarity in File Systems," *Proc. SIGMETRICS '98,* pp. 141-150, Apr. 1998.
[6] G. Zhang, J. Shu, W. Xue, and W. Zheng, "SLAS: An Efficient Approach to Scaling Round-Robin Striped Volumes," *ACM Trans. Storage ,* vol. 3, no. 1, pp. 1-39, Mar. 2007.
[7] H. Jin, X. Zhou, D. Feng, and J. Zhang, "Improving Partial Stripe Write Performance in RAID Level 5," *Proc. Second IEEE Int'l Caracas Conf. Devices, Circuits and Systems (ICCDCS '98),* pp. 396-400, Mar. 1998.
[8] A. Kuratti and W.H. Sanders, "Performance Analysis of the RAID 5 Disk Array," *Proc. Int'l Computer Performance and Dependability Symp.,* pp. 236-245, Apr. 1995.
[9] E. Lee and R. Katz, "The Performance of Parity Placements in Disk Arrays," *IEEE Trans. Computers,* vol. 42, no. 6, pp. 651-664, June 1993.
[10] C. Kim, G. Kim, and B. Shin, "Volume Management in SAN Environment," *Proc. Eighth Int'l Conf. Parallel and Distributed Systems (ICPADS '01),* pp. 500-505, 2001.
[11] P. Chen and E. Lee, "Striping in a RAID Level 5 Disk Array," *Proc. 1995 ACM SIGMETRICS Conf. Measurement and Modeling of Computer Systems,* May 1995.
[12] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* third ed. Morgan Kaufmann Publishers, Inc., 2003.
[13] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *ACM Trans. Computer Systems,* vol. 14, no. 1, pp. 108-136, Feb. 1996.
[14] D. Stodolsky, G. Gibson, and M. Holland, "Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays," *Proc. 20th Ann. Int'l Symp. Computer Architecture (ISCA '93),* pp. 64-75, 1993.
[15] A. Verma, U. Sharma, J. Rubas, D. Pease, M. Kaplan, R. Jain, M. Devarakonda, and M. Beigi, "An Architecture for Lifecycle Management in Very Large File Systems," *Proc. 22nd IEEE-13th NASA Goddard Conf. Mass Storage Systems and Technology (MSST '05),* Apr. 2005.
[16] C. Lu, G. Alvarez, and J. Wilkes, "Aqueduct: Online Data Migration with Performance Guarantees," *Proc. First USENIX Conf. File and Storage Technologies (FAST '02),* pp. 219-230, 2002.
[17] Performance Evaluation Laboratory, Trace Distribution Center, Brigham Young Univ., http://tds.cs.byu.edu/tds/, 2002.
[18] Public Software, Storage Systems Dept. at HP Labs, http://tesla.hpl.hp.com/public_software/, 2009.
[19] Storage Networking Industry Assoc., http://www.snia.org, 2009.
[20] L. Tian, D. Feng, H. Jiang, K. Zhou, L. Zeng, J. Chen, Z. Wang, and Z. Song, "PRO: A Popularity-Based Multi-Threaded Reconstruction Optimization for RAID-Structured Storage Systems," *Proc. Fifth USENIX Conf. File and Storage Technologies (FAST '07),* pp. 277-290, Feb. 2007.
[21] S. Jiang, X. Ding, F. Chen, E. Tan, and X. Zhang, "DULO: An Effective Buffer Cache Management Scheme to Exploit Both Temporal and Spatial Locality," *Proc. USENIX Conf. File and Storage Technologies (FAST '05),* Dec. 2005.
[22] C.B. Legg, "Method of Increasing the Storage Capacity of a Level Five RAID Disk Array by Adding, in a Single Step, a New Parity Block and N-1 New Data Blocks Which Respectively Reside in a new Columns, Where N Is at Least Two Document Type and Number," US Patent: 6000010, 1999.
[23] C.R Franklin and J.T. Wong, "Expansion of RAID Subsystems Using Spare Space with Immediate Access to New Space," US Patent 10/033,997, 2006.
[24] S.R. Hetzler, "Data Storage Array Scaling Method and System with Minimal Data Movement," US Patent 20080276057, 2008.
[25] M. Sivathanu, V. Prabhakaran, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Improving Storage System Availability with D-GRAID," *Proc. Third USENIX Conf. File and Storage Technologies,* Mar. 2004.
[26] M. Holland, G. Gibson, and D. Siewiorek, "Architectures and Algorithms for On-Line Failure Recovery in Redundant Disk Arrays," *Distributed and Parallel Databases,* vol. 11, no. 3, pp. 295-335, July 1994.
[27] J. Douceur and W. Bolosky, "Progress-Based Regulation of Low-Importance Processes," *Proc. 17th ACM Symp. Operating Systems Principles (SOSP '99),* pp. 247-260, Dec. 1999.
[28] A. Thomasian and J. Menon, "RAID5 Performance with Distributed Sparing," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 6, pp. 640-657, June 1997.
[29] G. Fu, A. Thomasian, C. Han, and S. Ng, "Rebuild Strategies for Redundant Disk Arrays," *Proc. Conf. Mass Storage Systems and Technologies (MSST '04),* Apr. 2004.
[30] R. Hou, J. Menon, and Y. Patt, "Balancing I/O Response Time and Disk Rebuild Time in a RAID5 Disk Array," *Proc. Hawaii Int'l Conf. System Sciences,* pp. 70-79, 1993.

Guangyan Zhang received the bachelor's and master's degrees in computer science from Jilin University in 2000 and 2003, respectively, and the doctor's degree in computer science and technology from Tsinghua University in 2008. He is now an assistant professor in the Department of Computer Science and Technology at Tsinghua University. His current research interests include network storage, parallel file systems, and distributed systems.

Weimin Zheng received the master's degree from Tsinghua University in 1982. He is a professor in the Department of Computer Science and Technology at Tsinghua University. His research interests include distributed computing, compiler techniques, and network storage.

Jiwu Shu received the doctor's degree in computer science from Nanjing University in 1998. In 2000, he finished his postdoctoral position research at Tsinghua University and has been teaching at Tsinghua University since then. He is now a professor in the Department of Computer Science and Technology at Tsinghua University. His current research interests include storage area networks, parallel and distributed computing.