FastScale: Accelerate RAID Scaling by Minimizing Data Migration

Weimin Zheng, Guangyan Zhang
Tsinghua University
gyzh@tsinghua.edu.cn

Abstract

Previous approaches to RAID scaling either require a very large amount of data to be migrated, or cannot tolerate multiple disk additions without resulting in disk imbalance. In this paper, we propose a new approach to RAID-0 scaling called FastScale. First, FastScale *minimizes data migration*, while maintaining a uniform data distribution. With a new and elastic addressing function, it moves only enough data blocks from old disks to fill an appropriate fraction of new disks without migrating data among old disks. Second, FastScale *optimizes data migration* with two techniques: (1) it accesses multiple physically successive blocks via a single I/O, and (2) it records data migration lazily to minimize the number of metadata writes without compromising data consistency. Using several real system disk traces, our experiments show that compared with SLAS, one of the most efficient traditional approaches, FastScale can reduce redistribution time by up to 86.06% with smaller maximum response time of user I/Os. The experiments also illustrate that the performance of the RAID-0 scaled using FastScale is almost identical with that of the round-robin RAID-0.

1 Introduction

Redundant Array of Inexpensive Disks (RAID) [1] was proposed to achieve high performance, large capacity and data reliability, while allowing a RAID volume to be managed as a single device. As user data increase and computing powers enhance, applications often require larger storage capacity and higher I/O performance. To supply needed capacity and/or bandwidth, one solution is to add new disks to a RAID volume. This disk addition is termed *"RAID scaling"*.

To regain uniform data distribution in all disks including the old and the new, RAID scaling requires certain blocks to be moved onto added disks. Furthermore, in today's server environments, many applications (e.g., e-business, scientific computation, and web servers) access data constantly. The cost of downtime is extremely high [2], giving rise to the necessity of online and real-time scaling.

Traditional approaches [3, 4, 5] to RAID scaling are restricted by preserving the round-robin order after adding disks. The addressing algorithm can be expressed as follows for the i^{th} scaling operation:

$$f_i(x): \begin{cases} d = x \bmod N_i \\ b = x/N_i \end{cases} \quad (1)$$

where block b of disk d is the location of logical block x, and N_i gives the total number of disks. Generally speaking, as far as RAID scaling from m disks to $m+n$ is concerned, only the data blocks in the first stripe are not moved. This indicates that almost 100 percent of data blocks have to be migrated no matter what the numbers of old disks and new disks are. There are some efforts [3, 5] concentrating on optimization of data migration. They improve the performance of RAID scaling by a certain degree, but do not overcome the limitation of large data migration completely.

The most intuitive method to reduce data migration is the semi-RR [6] algorithm. It requires a block movement only if the resulting disk number is one of new disks. The algorithm can be expressed as follows for the i^{th} scaling operation:

$$g_i(x) = \begin{cases} g_{i-1}(x) & \text{if } (x \bmod N_i) < N_{i-1} \\ f_i(x) & \text{otherwise} \end{cases} \quad (2)$$

Semi-RR reduces data migration significantly. Unfortunately, it does not guarantee uniform distribution of data blocks after subsequent scaling operations (see section 2.4). This will deteriorate the initial equally distributed load.

In this paper, we propose a novel approach called *FastScale* to redistribute data for RAID-0 scaling. It accelerates RAID-0 scaling by minimizing data migration.

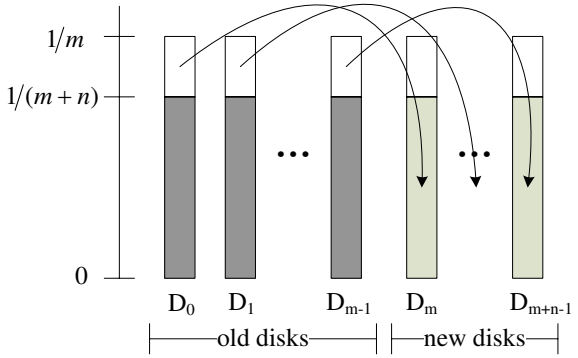

Figure 1: Data migration using FastScale. Only data blocks are moved from old disks to new disks for regaining a uniform distribution, while no data is migrated among old disks.

As shown in Figure 1, FastScale moves only data blocks from old disks to new disks enough for preserving the uniformity of data distribution, while not migrating data among old disks. Consequently, the migration fraction of FastScale reaches the lower bound of the migration fraction, $n/(m+n)$. In other words, FastScale succeeds in minimizing data migration for RAID scaling.

We design an elastic addressing function through which the location of one block can be easily computed without any lookup operation. By using this function, FastScale changes only a fraction of the data layout while preserving the uniformity of data distribution. FastScale has several unique features as follows:

- FastScale maintains a uniform data distribution after RAID scaling.
- FastScale minimizes the amount of data to be migrated entirely.
- FastScale preserves a simple management of data due to deterministic placement.
- FastScale can sustain the above three features after multiple disk additions.

FastScale also exploits special physical properties to optimize online data migration. First, it uses aggregate accesses to improve the efficiency of data migration. Second, it records data migration lazily to minimize the number of metadata updates while ensuring data consistency.

We implement a detailed simulator that uses DiskSim as a worker module to simulate disk accesses. Under several real-system workloads, we evaluate the traditional approach and the FastScale approach. The experimental results demonstrate that:

- Compared with one of the most efficient traditional approaches, FastScale shortens redistribution time by up to 86.06% with smaller maximum response time of user I/Os.
- The performance of the RAID scaled using FastScale is almost identical with that of the round-robin RAID.

In this paper, we only describe our solution for RAID-0, i.e., striping without parity. The solution can also work for RAID-10 and RAID-01. Although we do not handle RAID-4 and RAID-5, we believe that our method provides a good starting point for efficient scaling of RAID-4 and RAID-5 arrays.

2 Minimizing Data Migration

2.1 Problem Statement

For disk addition into a RAID, it is desirable to ensure an even load on all the disks and minimal block movement. Since the location of a block may be changed during a scaling operation, another objective is to quickly compute the current location of a block.

To achieve the above objectives, the following three requirements should be satisfied for RAID scaling:

- Requirement 1 (*Uniform data distribution*): If there are B blocks stored on m disks, the expected number of blocks on each disk is approximately B/m so as to maintain an even load.
- Requirement 2 (*Minimal Data Migration*): During the addition of n disks to a RAID with m disks storing B blocks, the expected number of blocks to be moved is $B \times n/(m+n)$.
- Requirement 3 (*Fast data Addressing*): In a m-disk RAID, the location of a block is computed by an algorithm with low space and time complexity.

2.2 Two Examples of RAID Scaling

Example 1: To understand how the FastScale algorithm works and how it satisfies all of the three requirements, we take RAID scaling from 3 disks to 5 as an example. As shown in Figure 2, one RAID scaling process can be divided into two stages logically: data migration and data filling. In the first stage, a fraction of existing data blocks are migrated to new disks. In the second stage, new data are filled into the RAID continuously. Actually, the two stages, data migration and data filling, can be overlapped in time.

For the RAID scaling, each 5 sequential locations in one disk are grouped into one *segment*. For the 5 disks, 5 segments with the same physical address are grouped

Figure 2: RAID scaling from 3 disks to 5 using FastScale, where $m \geq n$.

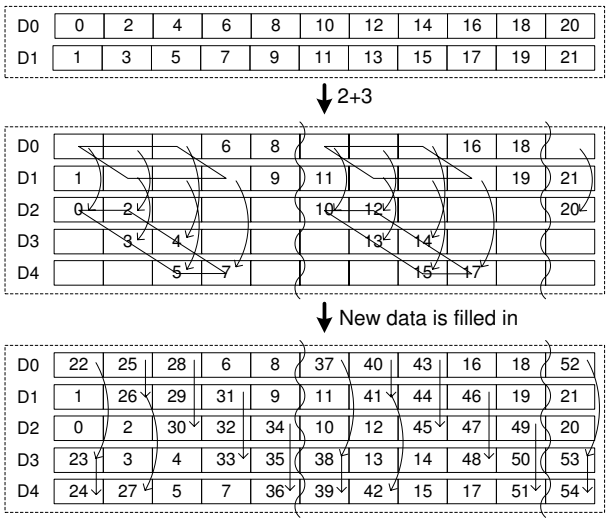

Figure 3: RAID scaling from 2 disks to 5 using FastScale, where $m < n$.

into one *region*. In Figure 2, different regions are separated with a wavy line. For different regions, the ways to data migration and data filling are exactly identical.

In a region, all of the data blocks within a parallelogram will be moved. The base of the parallelogram is 2, and the height is 3. In other words, 2 data blocks are selected from each old disk and migrated to new disks. The 2 blocks are sequential, and the start address is *disk_no*. Figure 2 depicts the moving trace of each migrating block. For one moving data block, only its physical disk number is changed while its physical block number is unchanged. As a result, the five columns of two new disks will contain 1, 2, 2, 1, and 0 migrated data blocks, respectively. Here, the data block in the first column will be placed upon disk 3, while the data block in the fourth column will be placed upon disk 4. The first blocks in columns 2 and 3 are placed on disk 3, and the second blocks in columns 2 and 3 are placed on disk 4. Thus, each new disk has 3 data blocks.

After data migration, each disk, either old or new, has 3 data blocks. That is to say, FastScale regains a uniform data distribution. The total number of data blocks to be moved is $2 \times 3 = 6$. This reaches the minimal number of moved blocks, $(5 \times 3) \times (2/(3+2)) = 6$. We can claim that the RAID scaling using FastScale can satisfy Requirement 1 and Requirement 2.

Let us examine whether FastScale can satisfy Requirement 3, i.e., fast data addressing. To consider how one logical data block is addressed, we divide all the data space in the RAID into three categories: original and unmoved data, original and migrated data, and new data. A conclusion can be drawn from the following description that the calculation overhead for the data addressing is very low.

- The original and unmoved data can be addressed with the original addressing method. In this example, the ordinal number of the disk holds one block x can be calculated: $d = x \mod 3$. Its physical block number can be calculated: $b = x/3$.

- The addressing method for original and migrated data can be obtained easily from the above description about the trace of the data migration. $b = x/3$. For those blocks in the first triangle, i.e., blocks 0, 3, and 4, we have $d = d_0 + 3$. For those blocks in the last triangle, i.e., blocks 7, 8, and 11, we have $d = d_0 + 2$. Here, d_0 is their original disk.

- Each region can hold $5 \times 2 = 10$ new blocks. In one region, how those new data blocks are placed is shown in Figure 2. If block x is a new block, it is the y^{th} new block, where $y = x - 3 \times 11$. Each stripe holds 2 new blocks. So, we have $b = y/2$. The first two new blocks in each region are placed on Blocks 0 of Disk 0 and 4. For the other blocks, $d = (y \mod 2) + (b \mod 5) - 1$.

Example 2: In the above example, the number of the old disks m and the number of the new disks n satisfy the condition: $m \geq n$. In the following, we inspect the case when $m < n$. Take RAID scaling from 2 disks to 5 as an example. Here, $m = 2$ and $n = 3$.

Likewise, in a region, all of the data blocks within a parallelogram will be moved. The base of the parallelogram is 3, and the height is 2. 3 consecutive data blocks are selected from each old disk and migrated to

Algorithm: Addressing(t, H, s, x, d, b)

Input:
t: scaling times
H: scaling history, H[0],..., H[t]
s: total number of data blocks in one disk
x: logical block number

Output:
d: the disk holding Block x
b: physical block number

```
1: if t = 0 then
2:       m ← H[0],    d ← x mod m,    b ← x / m
3:       exit
4: m ← H[t-1],    n ← H[t] - m,    δ ← m - H[0]
5: if x ∈ [0, m × s − 1]    // an original data block
6:       Addressing(t-1, H, s, x, d0, b0)
7:       b1 ← (b0 - δ) mod (m+n)
8:       if b1 ∈ [d0, d0 +n -1]        // to be moved
9:            d← Moving(d0, b1, m, n),    b←b0
10:      else        // not moved
11:           d←d0,    b←b0
12:else    // a new data block
13:      Placing(x, m, n, s, δ, d, b)
```

Table 1: The addressing algorithm using in FastScale.

Function: Moving(d_0, b_1, m, n)

Input:
d_0: the disk of the original location
b_1: the original location in a region
m: the number of old disks
n: the number of new disks

Output:
return value: new disk holding the block

```
1:  if m ≥ n
2:       if b1 ≤ n-1
3:            return d0+m
4:       if b1 ≥ m-1
5:            return d0+n
6:       return m+n-1- (b1-d0)
7:  if m < n
8:       if b1 ≤ m-1
9:            return d0+m
10:      if b1 ≥ n-1
11:           return d0+n
12:      return d0+ b1+1
```

Table 2: The Moving function.

new disks. Figure 3 depicts the trace of each migrating block. Similarly, for one moving data block, only its physical disk number is changed while its physical block number is unchanged. As a result, five columns of three new disks will have a different number of existing data blocks: 1, 2, 2, 1, 0. Here, the data block in the first column will be placed upon disk 3, while the data block in the fourth column will be placed upon disk 4. Unlike the first example, the first block in columns 2 and 3 are placed on disks 2 and 3, respectively. Thus, each new disk has 2 data blocks.

Similar to the first example, we can demonstrate that the RAID scaling using FastScale can satisfy the three requirements.

2.3 The Addressing Algorithm

Table 1 shows the algorithm to minimize data migration required by RAID scaling. The array H records the history of RAID scaling. *H[0]* is the initial number of disks in the RAID. After the i^{th} scaling operations, the RAID consists of *H[i]* disks.

When a RAID is constructed from scratch (i.e., $t = 0$), it is a round-robin RAID actually. The address of block x can be calculated via one division and one modular (line 2).

Let us inspect the t^{th} scaling, where n disks are added into a RAID made up of m disks (line 4).

(1) If block x is an original block (line 5), FastScale calculates its old address (d_0, b_0) before the t^{th} scaling (line 6).

- If (d_0, b_0) needs to be moved, FastScale changes the disk ordinal number while keeping the block ordinal number unchanged (line 9).
- If (d_0, b_0) does not need to be moved, FastScale keeps the disk ordinal number and the block ordinal number unchanged (line 11).

(2) If block x is a new block, FastScale places it via the Placing() procedure (line 13).

The code of line 8 is used to decide whether a data block (d_0, b_0) will be moved during RAID scaling. As shown in Figures 2 and 3, there is a parallelogram in each region. The base of the parallelogram is n, and the height is m. If and only if the data block is within a parallelogram, it will be moved. One parallelogram mapped to disk d_0 is a line segment. Its beginning and end are d_0 and $d_0 + n - 1$, respectively. If b_1 is within the line segment, block x is within the parallelogram, and therefore it will be moved. After a RAID scaling by adding n disks, the left-above vertex of the parallelogram proceeds by n blocks (line 7).

Once a data block is determined to be moved, FastScale changes its disk ordinal number with the Moving() function. As shown in Figure 4, a migrating parallelogram is divided into three parts: a head triangle, a body parallelogram, and a tail triangle. How a block moves depends on which part it lies in. No matter which is bigger between m and n, the head triangle and the tail

Procedure: Placing(x, m, n, s, δ, d, b)

Input:
x: logical block number
m: the number of old disks
n: the number of new disks
s: total number of data blocks in one disk
δ: offset of the first region

Output:
d: new disk holding the block
b: physical block of new location

```
1: y←x - m×s
2: b← y / n        row← y mod n
3: e← (b-δ) mod (m+n)
4: if    e < n
5:        if row < e+1
6:              d← row
7:        else
8:              d← row+m
9: else
10:       d← row+e-n+1
```

Table 3: The procedure to place new data.

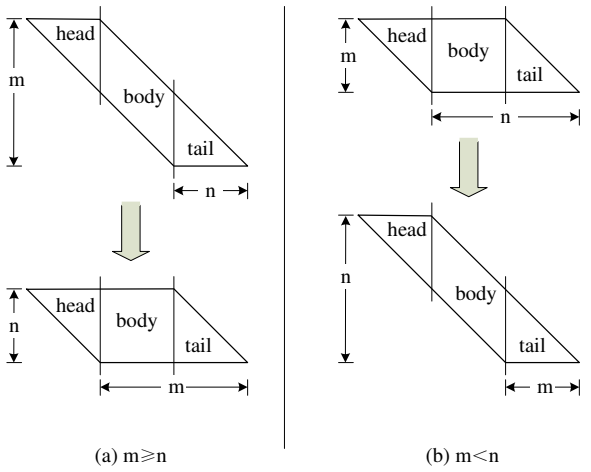

Figure 4: The variation of data layout involved in migration.

triangle keep their shapes unchanged. The head triangle will be moved by m disks (line 3, 9), while the tail triangle will be moved by n disks (line 5, 11). However, the body is sensitive to the relationship between m and n. The body is twisted from a parallelogram to a rectangle when $m \geq n$ (line 6), while from a rectangle to a parallelogram when $m < n$ (line 12). FastScale keeps the relative locations of all blocks in the same column.

When block x is in the location newly added after the last scaling, it is addressed via the Placing() procedure. If block x is a new block, it is the y^{th} new block (line 1). Each stripe holds n new blocks. So, we have $b = y/n$ (line 2). The order of placing new blocks is shown in Figures 2 and 3 (line 4-10).

This algorithm is very simple. It requires fewer than 50 lines of C code, reducing the likelihood that a bug will cause a data block to be mapped to the wrong location.

2.4 Property Examination

The purpose of this experiment is to quantitatively characterize whether the FastScale algorithm satisfies the three requirements, described in Subsection 2.1. For this purpose, we compare FastScale with the round-robin algorithm and the semi-RR algorithm. From a 4-disk array, we add one disk repeatedly for 10 times using the three algorithms respectively. Each disk has a capacity of 128 GB, and the size of a data block is 64 KB. In other words, each disk holds 2×1024^2 blocks.

Uniform data distribution. We use the coefficient of variation as a metric to evaluate the uniformity of data distribution across all the disks. The coefficient of variation expresses the standard deviation as a percentage of the average. The smaller the coefficient of variation is, the more uniform the data distribution is. Figure 5 plots the coefficient of variation versus the number of scaling operations. For the round-robin and FastScale algorithms, both the coefficients of variation remain 0 percent as the times of disk additions increases.

Conversely, the semi-RR algorithm causes excessive oscillation in the coefficient of variation. The maximum is even 13.06 percent. The reason for this non-uniformity is given as follows. An initial group of 4 disks makes the blocks be placed in a round-robin fashion. When the first scaling operation adds one disk, then $1/5$ of all blocks, where $(x \bmod 5) \geq 4$, are moved onto the new disk, Disk 4. However, with another operation of adding one more disk using the same approach, 1/6 of all the blocks are not evenly picked from the 5 old disks and moved onto the new disk, Disk 5. Only certain blocks from disks 1, 3 and 4 are moved onto disk 5 while disk 0 and disk 2 are ignored. This is because disk 5 will contain blocks with logical numbers that satisfy $(x \bmod 6) = 5$, which are all odd numbers. The logical numbers of those blocks on Disks 0 and 2, resulting from $(x \bmod 4) = 0$ and $(x \bmod 4) = 2$ respectively, are all even numbers. Therefore, blocks from disks 0 and 2 do not qualify and are not moved.

Minimal data migration. Figure 6 plots the migration fraction (i.e., the fraction of data blocks to be migrated) versus the number of scaling operations. Using the round-robin algorithm, the migration fraction is constantly 100%. This will bring a very large migration cost.

The migration fractions using the semi-RR algorithm and using FastScale are identical. They are significantly smaller than the migration fraction of using the round-

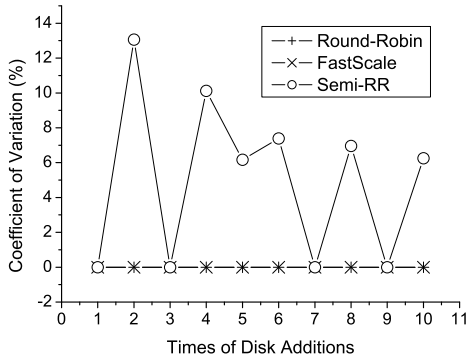

Figure 5: Comparison in uniform data distribution

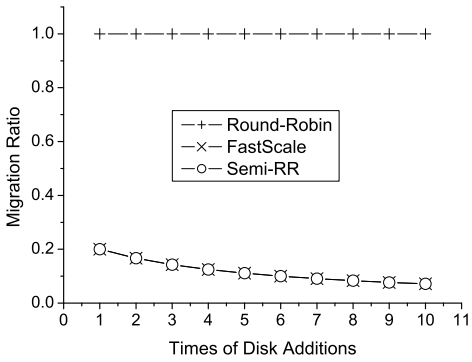

Figure 6: Comparison in data migration ratio

robin algorithm. Another obvious phenomenon is that they decrease with the increase of the number of scaling operations. The reason behind this phenomenon is described as follows. To make each new disk hold $1/(m+n)$ of total data, the semi-RR algorithm and FastScale moves $n/(m+n)$ of total data. m increases with the number of scaling operations. As a result, the percentage of new disks (i.e., $n/(m+n)$) decreases. Therefore, the migration fractions using the semi-RR algorithm and FastScale decrease.

Storage and calculation overheads. When a disk array boots, it needs to obtain the RAID topology from disks. Table 4 shows the storage overheads of the three algorithms. The round-robin algorithm depends only on the total number of member disks. So its storage overhead is one integer. The semi-RR and FastScale algorithms depend on how many disks are added during each scaling operation. If we scale RAID t times, their storage overheads are t integers. Actually, the RAID scaling operation is not too frequent. It may be performed every half year, or even longer. Consequently, the storage overheads are very small.

To quantitatively characterize the calculation overheads, we run different algorithms to calculate the physical addresses for all data blocks on a scaled RAID. The whole addressing process is timed and then the average addressing time for each block is calculated. The testbed used in the experiment is an Intel Dual Core T9400 2.53 GHz machine with 4 GB of memory. A Windows 7 Enterprise Edition is installed. Figure 7 plots the addressing time versus the number of scaling operations.

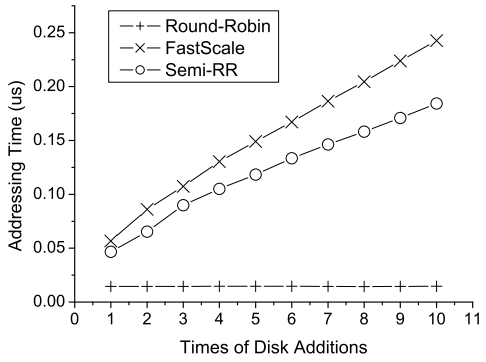

Figure 7: Comparison in addressing time

Algorithm	Storage Overhead
round-robin	1
semi-RR	t
FastScale	t

Table 4: The storage overheads of different algorithms.

The round-robin algorithm has a low calculation overhead of 0.014 μs or so. The calculation overheads using the semi-RR and FastScale algorithms are close, and both take on an upward trend. Among the three algorithms, FastScale has the largest overhead. Fortunately, the largest addressing time using FastScale is 0.24 μs which is negligible compared to milliseconds of disk I/O time.

3 Optimizing Data Migration

The FastScale algorithm succeeds in minimizing data migration for RAID scaling. In this section, we describe FastScale's optimizations to the process of data migration.

3.1 Access Aggregation

FastScale moves only data blocks from old disks to new disks, while not migrating data among old disks. The data migration will not overwrite any valid data. As a result, data blocks may be moved in an arbitrary order. Since disk I/O performs much better with large sequential access, FastScale accesses multiple successive blocks via a single I/O.

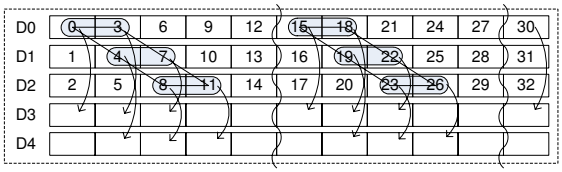

Figure 8: Aggregate reads for RAID scaling from 3 disks to 5. Multiple successive blocks are read via a single I/O.

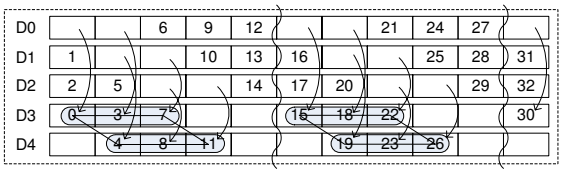

Figure 9: Aggregate writes for RAID scaling from 3 disks to 5. Multiple successive blocks are written via a single I/O.

Take a RAID scaling from 3 disks to 5 as an example, shown in Figure 8. Let us focus on the first region. FastScale issues the first I/O request to read Blocks 0 and 3, the second request to read Blocks 4 and 7, and the third request for Blocks 8 and 11, simultaneously. By this means, to read all of these blocks, FastScale requires only three I/Os, instead of six. Furthermore, all these 3 large-size data reads are on three disks. They can be done in parallel, further increasing I/O rate.

When all the six blocks have been read into a memory buffer, FastScale issues the first I/O request to write Blocks 0, 3, and 7, the second I/O to write Blocks 4, 8 and 11, simultaneously (see Figure 9). In this way, only two large sequential write requests are issued as opposed to six small writes.

For RAID scaling from m disks to $m+n$, m reads and n writes are required to migrate all the data in a region, i.e., $m \times n$ data blocks.

Access aggregation converts sequences of small requests into fewer, larger requests. As a result, seek cost is mitigated over multiple blocks. Moreover, a typical choice of the optimal block size for RAID is 32KB or 64KB [4, 7, 8, 9]. Thus, accessing multiple successive blocks via a single I/O enables FastScale to have a larger throughput. Since data densities in disks increase at a much faster rate than improvements in seek times and rotational speeds, access aggregation benefits more as technology advances.

3.2 Lazy Checkpoint

While data migration is in progress, the RAID storage serves user requests. Furthermore, the coming user I/Os may be write requests to migrated data. As a result, if mapping metadata does not get updated until all of the blocks have been moved, data consistency may be destroyed. Ordered operations [9] of copying a data

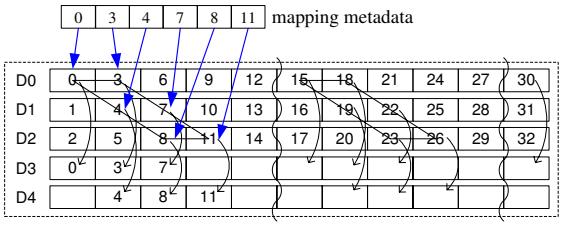

Figure 10: If data blocks are copied to their new locations and metadata is not yet updated when the system fails, data consistency is still maintained because the data in their original locations are valid and available.

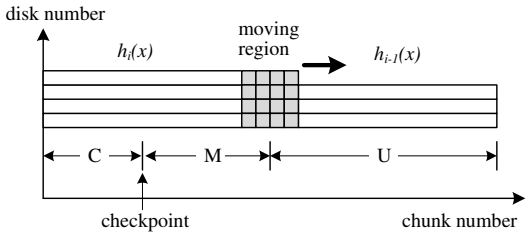

Figure 11: Lazy updates of mapping metadata. "C": migrated and checkpointed; "M": migrated but not checkpointed; "U":not migrated. Data redistribution is checkpointed only when a user write request arrives in the area "M".

block and updating the mapping metadata (a.k.a., *checkpoint*) can ensure data consistency. But ordered operations cause each block movement to require one metadata write, which results in a large cost of data migration. Because metadata is usually stored at the beginning of all member disks, each metadata update causes one long seek per disk. FastScale uses lazy checkpoint to minimize the number of metadata writes without compromising data consistency.

The foundation of lazy checkpoint is described as follows. Since block copying does not overwrite any valid data, both its new replica and original are valid after a data block is copied. In the above example, we suppose that Blocks 0, 3, 4, 7, 8, and 11 have been copied to their new locations and the mapping metadata has not been updated (see Figure 10), when the system fails. The original replicas of the six blocks will be used after the system reboots. As long as Blocks 0, 3, 4, 7, 8, and 11 have not been written since being copied, the data remain consistent. Generally speaking, when the mapping information is not updated immediately after a data block is copied, an unexpected system failure only wastes some data accesses, but does not sacrifice data reliability. The only threat is the incoming of write operations to migrated data.

The key idea behind lazy checkpoint is that data blocks are copied to new locations continuously, while the mapping metadata is not updated onto the disks (a.k.a., *checkpoint*) until a threat to data consistency appears. We use $h_i(x)$ to describe the geometry after the i^{th} scaling opera-

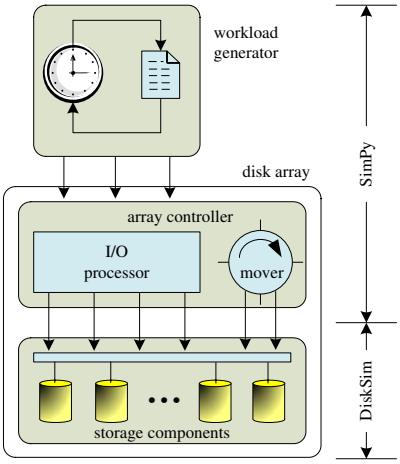

Figure 12: Simulation system block diagram: The workload generator and the array controller were implemented in SimPy. DiskSim was used as a worker module to simulate disk accesses.

tion, where N_i disks serve user requests. Figure 11 illustrates an overview of the migration process. Data in the moving region is copied to new locations. When a user request arrives, if its physical block address is above the moving region, it is mapped with $h_{i-1}(x)$; If its physical block address is below the moving region, it is mapped with $h_i(x)$. When all of the data in the current moving region are moved, the next region becomes the moving region. In this way, the newly added disks are gradually available to serve user requests. Only when a user write request arrives in the area where data have been moved and the movement has not been checkpointed, are mapping metadata updated.

Since one write of metadata can store multiple map changes of data blocks, lazy updates can significantly decrease the number of metadata updates, reducing the cost of data migration. Furthermore, lazy checkpoint can guarantee data consistency. Even if the system fails unexpectedly, only some data accesses are wasted. It should also be noted that the probability of a system failure is very low.

4 Experimental Evaluation

The experimental results in Section 2.4 show that the semi-RR algorithm causes extremely non-uniform data distribution. This will bring into low I/O performance. In this section, we compare FastScale with the SLAS approach [5] through detailed experiments. SLAS, proposed in 2007, preserves the round-robin order after adding disks.

4.1 Simulation System

We use detailed simulations with several disk traces collected in real systems. The simulator is made up of a workload generator and a disk array (Figure 12). 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 disk array.

The disk array consists of an array controller and storage components. The array controller is logically divided into two parts: an I/O processor and a data mover. The I/O processor, according to the address mapping, forwards incoming I/O requests to the corresponding disks. The data mover reorganizes the data on the array. The mover uses an on/off logic to adjust the redistribution rate. Data redistribution is throttled on detection of high application workload. Otherwise, it performs continuously.

The simulator is implemented in SimPy [10] and DiskSim [11]. SimPy is an object-oriented, process-based discrete-event simulation language based on standard Python. DiskSim is an efficient, accurate disk system simulator from Carnegie Mellon University and has been extensively used in various research projects studying storage subsystem architectures. The workload generator and the array controller are implemented in SimPy. Storage components are implemented in DiskSim. In other words, DiskSim is used as a worker module to simulate disk accesses. The simulated disk specification is that of the 15,000-RPM IBM Ultrastar 36Z15 [12].

4.2 Workloads

Our experiments use 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 [13]. It was collected with one client running 20 iterations.
- *Fin* is obtained from the Storage Performance Council (SPC) [14, 15], a vendor-neutral standards body. The Fin trace was collected from OLTP applications running at a large financial institution. The write ratio is high.
- *Web* is also from SPC. It was collected from a system running a web search engine. The read-dominated Web trace exhibits the strong locality in its access pattern.

4.3 Experiment Results

4.3.1 The Scaling Efficiency

Each experiment lasts from the beginning to the end of data redistribution for RAID scaling. We focus on com-

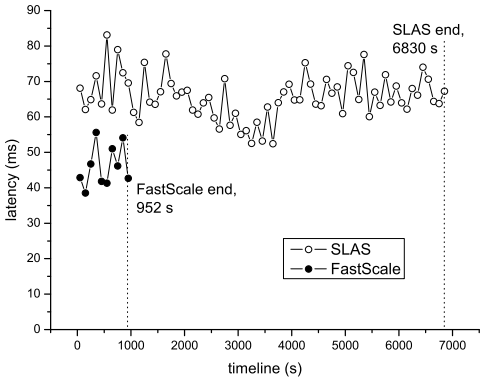

Figure 13: Performance comparison between FastScale and SLAS under the Fin workload.

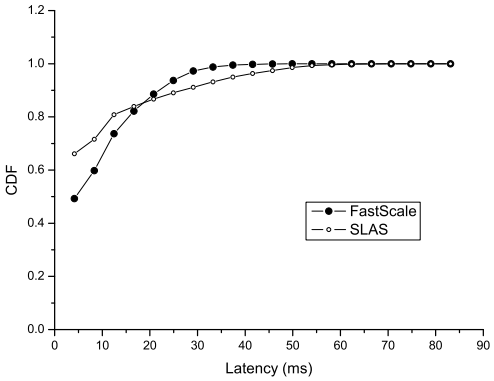

Figure 14: Cumulative distribution of I/O latencies during the data redistributions by the two approaches under the Fin workload.

paring redistribution times and user I/O latencies when different scaling programs are running in background.

In all experiments, the sliding window size for SLAS is set to 1024. Access aggregation in SLAS can improve the redistribution efficiency. However, a too large size of redistribution I/Os will compromise the I/O performance of applications. In our experiments, SLAS reads 8 data blocks via an I/O request.

The purpose of our first experiment is to quantitatively characterize the advantages of FastScale through a comparison with SLAS. We conduct a scaling operation of adding 2 disks to a 4-disk RAID, where each disk has a capacity of 4 GB. Each approach performs with the 32KB stripe unit size under a Fin workload. The threshold of rate control is set 100 IOPS. This parameter setup acts as the baseline for the latter experiments, from which any change will be stated explicitly.

We collect the latencies of all user I/Os. We divide the I/O latency sequence into multiple sections according to I/O issuing time. The time period of each section is 100 seconds. Furthermore, we get a local maximum latency from each section. A local maximum latency is the maximum of I/O latencies in a section. Figure 13 plots local maximum latencies using the two approaches as the time increases along the x-axis. It illustrates that FastScale demonstrates a noticeable improvement over SLAS in two metrics. First, the redistribution time using FastScale is significantly shorter than that using SLAS. They are 952 seconds and 6,830 seconds, respectively. In other words, FastScale has a 86.06% shorter redistribution time than SLAS.

The main factor in FastScale's reducing the redistribution time is the significant decline of the amount of the data to be moved. When SLAS is used, almost 100% of data blocks have to be migrated. However, when FastScale is used, only 33.3% of data blocks require to be migrated. Another factor is the effective exploitation of two optimization technologies: access aggregation reduces the number of redistribution I/Os; lazy checkpoint minimizes metadata writes.

Second, local maximum latencies of SLAS are obviously longer than those of FastScale. The global maximum latency using SLAS reaches 83.12 ms while that using FastScale is 55.60 ms. This is because the redistribution I/O size using SLAS is larger than that using FastScale. For SLAS, the read size is 256 KB (8 blocks), and the write size is 192 KB (6 blocks). For FastScale, the read size is 64 KB (2 blocks), and the write size is 128 KB (4 blocks). Of course, local maximum latencies of SLAS will be lower with a decrease in the redistribution I/O size. But the decrease in the I/O size will necessarily enlarge the redistribution time.

Figure 14 shows the cumulative distribution of user response times during data redistribution. To provide a fair comparison, I/Os involved in statistics for SLAS are only those issued before 952 seconds. When I/O latencies are larger than 18.65 ms, the CDF value of FastScale is larger than that of SLAS. This indicates again that FastScale has smaller maximum response time of user I/Os than SLAS. The average latency of FastScale is close to that of SLAS. They are 8.01 ms and 7.53 ms respectively. It is noteworthy that due to significantly shorter data redistribution time, FastScale has a markedly smaller impact on the user I/O latencies than SLAS does.

A factor that might affect the benefits of FastScale is the workload under which data redistribution performs. Under the TPC-C workload, we also measure the performances of FastScale and SLAS to perform the "4+2" scaling operation.

For the TPC-C workload, Figure 15 shows local maximum latencies versus the redistribution times for SLAS and FastScale. It shows once again the efficiency of FastScale in improving the redistribution time. The redistribution times using SLAS and FastScale are 6,820 seconds and 964 seconds, respectively. That is to say, FastScale brings an improvement of 85.87% in the re-

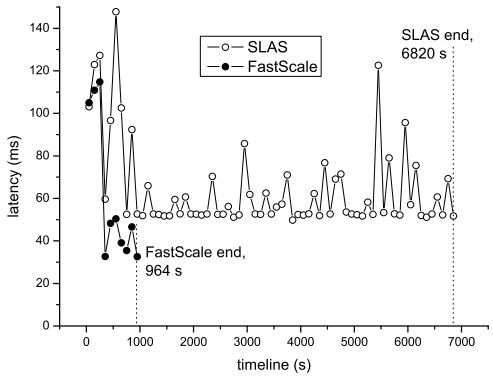

Figure 15: Performance comparison between FastScale and SLAS under the TPC-C workload.

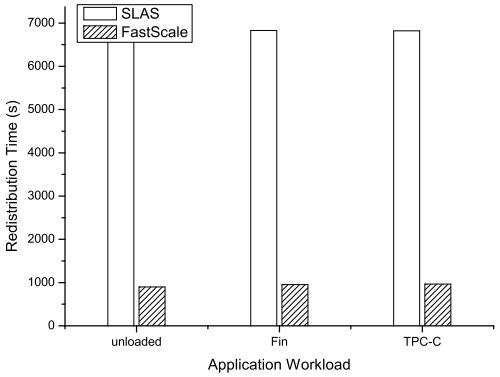

Figure 16: Comparison of redistribution times of FastScale and SLAS under different workloads. The label "unloaded" means scaling a RAID volume offline.

distribution time. Likewise, local maximum latencies of FastScale are also obviously shorter than those of SLAS. The global maximum latency using FastScale is 114.76 ms while that using SLAS reaches 147.82 ms.

To compare the performance of FastScale under different workloads, Figure 16 shows a comparison in the redistribution time between FastScale and SLAS. For completeness, we also conducte a comparison experiment on the redistribution time with no loaded workload. To scale a RAID volume off-line, SLAS uses 6802 seconds whereas FastScale consumes only 901 seconds. FastScale provides an improvement of 86.75% in the redistribution time.

We can draw one conclusion from Figure 16. Under various workloads, FastScale consistently outperformes SLAS by 85.87-86.75% in the redistribution time, with smaller maximum response time of user I/Os.

4.3.2 The Performance after Scaling

The above experiments show that FastScale improves the scaling efficiency of RAID significantly. One of our concerns is whether there is a penalty in the performance of the data layout after scaling using FastScale, compared with the round-robin layout preserved by SLAS.

We use the Web workload to measure the performances of the two RAIDs, scaled from the same RAID using SLAS and FastScale. Each experiment lasts 500 seconds, and records the latency of each I/O. Based on the issue time, the I/O latency sequence is divided into 20 sections evenly. Furthermore, we get a local average latency from each section.

First, we compare the performances of two RAIDs, after one scaling operation "4+1" using the two scaling approaches. Figure 17 plots local average latencies for the two RAIDs as the time increases along the x-axis. We can find that the performances of the two RAIDs are very close. With regards to the round-robin RAID, the average latency is 11.36 ms. For the FastScale RAID, the average latency is 11.37 ms.

Second, we compare the performances of two RAIDs, after two scaling operations "4+1+1" using the two approaches. Figure 18 plots local average latencies of the two RAIDs as the time increases along the x-axis. It again revealed the approximate equality in the performances of the two RAIDs. With regards to the round-robin RAID, the average latency is 11.21 ms. For the FastScale RAID, the average latency is 11.03 ms.

One conclusion can be reached that the performance of the RAID scaled using FastScale is almost identical with that of the round-robin RAID.

5 Related Work

5.1 Scaling Deterministic RAID

The HP AutoRAID [8] allows an online capacity expansion. 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. This expansion does not require data migration. But the system cannot add new disks into an existing RAID-5 volume. The conventional approaches to RAID scaling redistributes data and preserves the round-robin order after adding disks.

Gonzalez and Cortes [3] proposed a gradual assimilation algorithm (GA) to control the overhead of scaling a RAID-5 volume. However, GA accesses only one block via an I/O. Moreover, it writes mapping metadata onto disks immediately after redistributing each stripe. As a result, GA has a large redistribution cost.

The reshape toolkit in the Linux MD driver (MD-Reshape) [4] writes mapping metadata for each fixed-sized data window. However, user requests to the data window have to queue up until all data blocks within the window are moved. On the other hand, MD-Reshape issues very small (4KB) I/O operations for data redistribution. This limits the redistribution performance due to

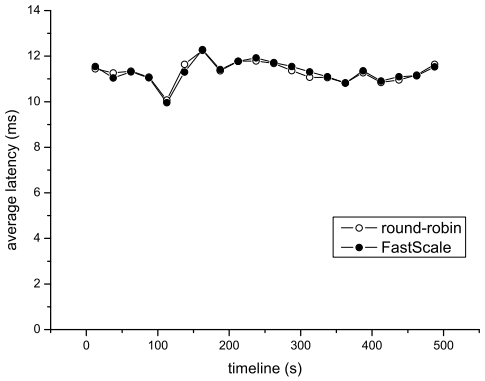

Figure 17: Performance comparison between FastScale's layout and round-robin layout under the Web workload after one scaling operation "4+1".

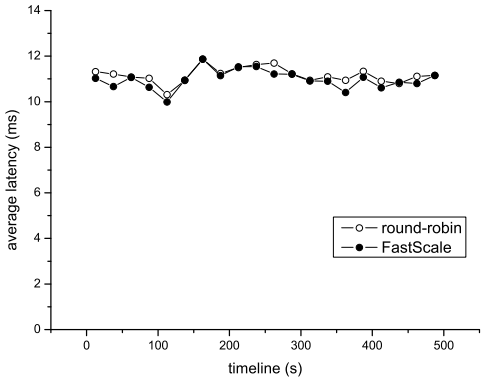

Figure 18: Performance comparison between FastScale's layout and round-robin layout under the Web workload after two scaling operations "4+1+1".

more disk seeks.

Zhang et al. [5] discovered that there is always a reordering window during data redistribution for round-robin RAID scaling. The data inside the reordering window can migrate in any order without overwriting any valid data. By leveraging this insight, they proposed the SLAS approach, improving the efficiency of data redistribution. However, SLAS still requires migrating all data. Therefore, RAID scaling remains costly.

D-GRAID [16] 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, FastScale is independent on file systems, and it can work with any ordinary disk storage.

A patent [17] presents a method to eliminate the need to rewrite the original data blocks and parity blocks on original disks. However, the method makes all the parity blocks be either only on original disks or only on new disks. The obvious distribution non-uniformity of parity blocks will bring a penalty to write performance.

Franklin et al. [18] presented an RAID scaling method using spare space with immediate access to new space. First, 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, new data are copied from the spare space to the set of data disk drives. This is similar to the key idea of WorkOut [19]. This kind of method requires spare disks available in the RAID.

In another patent, Hetzler [20] presented a method to RAID-5 scaling, noted MDM. MDM exchanges some data blocks between original disks and new disks. MDM can perform RAID scaling with reduced data movement. However, it does not increase (just maintains) the data storage efficiency after scaling. The RAID scaling process exploited by FastScale is favored in the art because the data storage efficiency is maximized, which many practitioners consider desirable.

5.2 Scaling Randomized RAID

Randomized RAID [6, 21, 22, 23] appears to have better scalability. It is now gaining the spotlight in the data placement area. Brinkmann et al. [23] proposed the cut-and-paste placement strategy that uses randomized allocation strategy to place data across disks. For a disk addition, it cuts off the range $[1/(n+1), 1/n]$ from given n disks, and pastes them to the newly added $(n+1)^{th}$ disk. For a disk removal, it uses reversing operation to move all the blocks in disks that will be removed to the other disks. Also based on random data placement, Seo and Zimmermann [24] proposed an approach to finding a sequence of disk additions and removals for the disk replacement problem. The goal is to minimize the data migration cost. Both these two approaches assume the existence of a high-quality hash function that assigns all the data blocks in the system into the uniformly distributed real numbers with high probability. However, they did not present such a hash function.

The SCADDAR algorithm [6] uses a pseudo-random function to distribute data blocks randomly across all disks. It keeps track of the locations of data blocks after multiple disk reorganizations and minimizes the amount of data to be moved. Unfortunately, the pseudo-hash function does not preserve the randomness of the data layout after several disk additions or deletions [24]. So far, true randomized hash function which preserves its randomness after several disk additions or deletions has not been found.

The simulation report in [21] shows that a single copy of data in random striping may result in some hiccups of the continuous display. To address this issue, one can use data replication [22], where a fraction of the data blocks

randomly selected are replicated on randomly selected disks. However, this will bring into a large capacity overhead.

RUSH [25, 26] and CRUSH [27] are two algorithms for online placement and reorganization of replicated data. They are probabilistically optimal in distributing data evenly and minimizing data movement when new storage is added to the system. There are three differences between them and FastScale. First, they depend on the existence of a high-quality random function, which is difficult to generate. Second, they are designed for object-based storage systems. They focus on how a data object is mapped to a disk, without considering the data layout of each individual disk. Third, our mapping function needs to be 1-1 and onto, but hash functions have collisions and count on some amount of sparseness.

6 Conclusion and Future Work

This paper presents FastScale, a new approach that accelerates RAID-0 scaling by minimizing data migration. First, with a new and elastic addressing function, FastScale minimizes the number of data blocks to be migrated without compromising the uniformity of data distribution. Second, FastScale uses access aggregation and lazy checkpoint to optimize data migration.

Our results from detailed experiments using real-system workloads show that, compared with SLAS, a scaling approach proposed in 2007, FastScale can reduce redistribution time by up to 86.06% with smaller maximum response time of user I/Os. The experiments also illustrate that the performance of the RAID scaled using FastScale is almost identical with that of the round-robin RAID.

In this paper, the factor of data parity is not taken into account. we believe that FastScale provides a good starting point for efficient scaling of RAID-4 and RAID-5 arrays. In the future, we will focus on extending FastScale to RAID-4 and RAID-5.

7 Acknowledgements

We are indebted to the anonymous reviewers of the paper for their insightful comments. We are also grateful to Dr. Benjamin Reed, our shepherd, for detailed comments and suggestions that greatly improved the readability of the paper. This work was supported by the National Natural Science Foundation of China under Grant 60903183, the National High Technology Research and Development Program of China under Grant No. 2009AA01A403, and the National Grand Fundamental Research 973 Program of China under Grant No. 2007CB311100.

References

[1] D. A. Patterson, G. A. Gibson, R. H. Katz. A Case for Redundant Arrays of Inexpensive Disks (RAID), in Proceedings of the International Conference on Management of Data (SIGMOD'88), June 1988. pp. 109-116.

[2] D. A. Patterson. A simple way to estimate the cost of down-time. In Proceedings of the 16th Large Installation Systems Administration Conference (LISA'02), October 2002. pp. 185-188.

[3] J. Gonzalez and T. Cortes. Increasing the capacity of RAID5 by online gradual assimilation. In Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os. Antibes Juan-les-pins, France, Sept. 2004

[4] N. Brown. Online RAID-5 resizing. drivers/md/ raid5.c in the source code of Linux Kernel 2.6.18. http://www.kernel.org/. September 2006.

[5] G. Zhang, J. Shu, W. Xue, and W. Zheng. SLAS: An efficient approach to scaling round-robin striped volumes. ACM Trans. Storage, volume 3, issue 1, Article 3, 1-39 pages. March 2007.

[6] A. Goel, C. Shahabi, S-YD Yao, R. Zimmermann. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In Proceedings of the 18th International Conference on Data Engineering (ICDE'02). San Jose, 2002. pp. 473-482.

[7] J. Hennessy and D. Patterson. Computer Architecture: A Quantitative Approach, 3rd ed. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 2003.

[8] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. ACM Transactions on Computer Systems, volume 14, issue 1, pp. 108-136, February 1996.

[9] C. Kim, G. Kim, and B. Shin. Volume management in SAN environment. In Proceedings of the 8th International Conference on Parallel and Distributed Systems, ICPADS'01. 2001. pp. 500-505.

[10] Klaus Muller, Tony Vignaux. SimPy 2.0.1's documentation. http://simpy.sourceforge.net/SimPyDocs/index.html. last accessed on April, 2009.

[11] J. Bucy, J. Schindler, S. Schlosser, G. Ganger. The DiskSim Simulation Environment Version 4.0 Reference Manual. Tech. report CMU-PDL-08-101, Carnegie Mellon University. 2008.

[12] Hard disk drive specifications Ultrastar 36Z15. http://www.hitachigst.com/tech/techlib.nsf/techdocs/85256AB8006A31E587256A7800739FEB/$file/U36Z15_sp10.PDF. Revision 1.0, April, 2001.

[13] TPC-C. Postgres. 20 iterations. DTB v1.1. Performance Evaluation Laboratory, Brigham Young University. Trace distribution center. http://tds.cs.byu.edu/tds/, last accessed on December, 2010.

[14] OLTP Application I/O and Search Engine I/O. UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage. June, 2007.

[15] Storage Performance Council. http://www.storageperformance.org/home. last accessed on December, 2010.

[16] Muthian Sivathanu , Vijayan Prabhakaran , Andrea C. Arpaci-Dusseau , Remzi H. Arpaci-Dusseau. Improving Storage System Availability with D-GRAID, In Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST'04), San Francisco, CA. March 2004.

[17] 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, US Patent: 6000010, December 1999.

[18] 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.

[19] Suzhen Wu, Hong Jiang, Dan Feng, Lei Tian, and Bo Mao, WorkOut: I/O Workload Outsourcing for Boosting the RAID Reconstruction Performance, In Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST '09), San Francisco, CA, USA, pp. 239-252. February 2009.

[20] S.R. Hetzler, Data Storage Array Scaling Method and System with Minimal Data Movement, US Patent 20080276057, 2008.

[21] J. Alemany and J. S. Thathachar. Random striping news on demand servers. Tech. Report, TR-97-02-02, University of Washington, 1997.

[22] Jose Renato Santos, Richard R. Muntz, and Berthier A. Ribeiro-Neto. Comparing random data allocation and data striping in multimedia servers. In Measurement and Modeling of Computer Systems, pp. 44-55. 2000.

[23] Andre Brinkmann, Kay Salzwedel, and Christian Scheideler. Efficient, distributed data placement strategies for storage area networks (extended abstract). In ACM Symposium on Parallel Algorithms and Architectures, pp. 119-128. 2000.

[24] Beomjoo Seo and Roger Zimmermann. Efficient disk replacement and data migration algorithms for large disk subsystems. ACM Transactions on Storage (TOS), volume 1, issue 3, pages 316-345, August 2005.

[25] R. J. Honicky and E. L. Miller. A fast algorithm for online placement and reorganization of replicated data. In Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), Nice, France, April 2003.

[26] R. J. Honicky and E. L. Miller. Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution. In Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE. 2004

[27] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn. CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In Proceedings of the International Conference on Super Computing (SC'06). Tampa Bay, FL. 2006.