# Redistribute Data to Regain Load Balance during RAID-4 Scaling

Guangyan Zhang, Jigang Wang, Keqin Li, Jiwu Shu, *Member, IEEE*, and Weimin Zheng

**Abstract**—Disk additions to a RAID-4 storage system can increase the I/O parallelism and expand the storage capacity simultaneously. To regain load balance among all disks including old and new, RAID-4 scaling requires moving certain data blocks onto newly added disks. Existing data redistribution approaches to RAID-4 scaling, restricted by preserving a round-robin data distribution, require migrating all the data, which results in an expensive cost for RAID-4 scaling. In this paper, we propose McPod—a new data redistribution approach to accelerating RAID-4 scaling. McPod minimizes the number of data blocks to be moved while maintaining a uniform data distribution across all data disks. McPod also optimizes data migration with four techniques. First, it coalesces multiple accesses to physically successive blocks into a single I/O. Second, it piggybacks parity updates during data migration to reduce the cost of maintaining consistent parities. Third, it outsources all parity updates brought by RAID scaling to a surrogate disk. Fourth, it delays recording data migration on disks to minimize the number of metadata writes without compromising data reliability. We implement McPod in Linux Kernel 2.6.32.9, and evaluate its performance by replaying three real-system traces. The results demonstrate that McPod outperforms the existing "moving-everything" approach by 67.78-79.64 percent in redistribution time and by 14.24-27.16 percent in user response time. The experiments also illustrate that the performance of the RAID scaled using McPod is almost identical to that of the round-robin RAID.

**Index Terms**—Access coalescing, data migration, I/O parallelism, metadata update, parity update, RAID-4 scaling

## 1 Introduction

### 1.1 Motivation

RAID [1], [2] achieves high performance, large capacity, and fault tolerance via disk striping and rotated parity. When the capacity or the bandwidth of a RAID system is insufficient [3], [4], more disks may be added. In order to regain load balance, data need to be redistributed among all disks. In today's server environments where applications access data constantly, the cost of downtime is extremely high [5]. Therefore, data redistribution needs to be performed online. Such disk addition is termed *RAID scaling*.

Standard RAID levels related to this paper include 0, 4, 5, and 6. These RAID levels have obviously different data layouts. Therefore, those efficient scaling approaches, developed for RAID-0 [6], [7], RAID-5 [8], [9], and RAID-6 [10], are not suitable for RAID-4. An efficient scaling approach should be designed based on the characteristics of RAID 4. RAID-4 tolerates single disk failures by maintaining parity on its dedicated parity disk. While the parity disk tends to become a write performance bottleneck, read performance is potentially fast because data can be read in parallel. A solution like this might be ideal for applications that are read intensive. Video-on-demand is a good example of such a situation. Furthermore, the NetApp WAFL file system [11] works with RAID-4, where the write performance bottleneck is alleviated by buffering writes in memory and then writing full RAID stripes.

Existing approaches [12], [13], [14] to RAID-4 scaling are restricted by preserving a round-robin data distribution. One of the obvious advantages is that load balance among all the disks can be regained due to the uniform data distribution. However, 100 percent of data blocks have to be migrated, which will incur a very large cost of RAID-4 scaling. This cost includes data migration, and metadata updates for keeping track of such data migration. During RAID scaling, data redistribution and foreground applications share and even contend for I/O resources in the system. The expensive cost for RAID-4 scaling means that either data redistribution will be completed in a long time, or the impact of data redistribution on application performance will be significant. There are some efforts [12], [14] concentrating on optimization of data migration for RAID scaling. They can be used to improve the performance of RAID-4 scaling to certain extent, but cannot completely solve the problem of large data migration.

### 1.2 Technical Challenges

Zheng and Zhang [6], [7] proposed the FastScale approach to accelerate RAID-0 scaling by minimizing data migration. FastScale provides a good starting point for efficient scaling of RAID-4 arrays. However, optimizing data redistribution for RAID-4 scaling will be more difficult, due to maintaining consistent data parities.

First, write operations are indispensable for data redistribution. The parity disk tends to become a performance


- *G. Zhang, J. Shu, and W. Zheng are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {gyzh, shujw, zwm-dcs}@tsinghua.edu.cn.*
- *J. Wang is with the Integration of Product R&D, ZTE corporation, Nanjing 210012, China. E-mail: wang.jigang@zte.com.cn.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, New York 12561. E-mail: lik@newpaltz.edu.*


*Manuscript received 5 Nov. 2013; revised 25 Dec. 2013; accepted 30 Dec. 2013. Date of publication 24 Feb. 2014; date of current version 5 Dec. 2014. Recommended for acceptance by D. Simplot-Ryl.*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TPDS.2014.2308219*

bottleneck. How to eliminate this performance bottleneck during RAID-4 scaling is a new technical challenge. Second, optimization of metadata updates in FastScale only needs to guarantee data consistency. During RAID-4 scaling, it is still required to maintain the competence of tolerating one disk failure. How to optimize metadata updates, without loss of data consistency and data reliability, is a new technical challenge.

RAID-4 scaling requires an efficient approach to redistributing data online, satisfying the following six requirements.

- After RAID-4 scaling, data are distributed evenly across all data disks including old disks and new.
- During a scaling operation, the amount of data to be migrated is minimal.
- The location of a block can be computed easily without any lookup operation.
- Data consistency can be guaranteed even if the system crashes during the scaling process.
- Data will not be lost even if one disk fails in the middle of scaling.
- The above five features can still be sustained after multiple disk additions.

### 1.3 Our Contributions

In this paper, we propose McPod (*M*inimizing data migration, *c*oalesced I/O operations, *P*iggyback parity updates, *O*utsourcing parity updates, and *D*elayed metadata updates)—a new data redistribution approach to accelerating RAID-4 scaling. McPod moves data blocks from old disks to new disks just enough to preserve the uniformity of data distribution. The migration fraction of McPod reaches the lower bound of the migration fraction, with the restriction of maintaining a uniform data distribution.

McPod also optimizes online data migration with four techniques. First, it coalesces multiple accesses to physically successive blocks into a single I/O. Second, it piggybacks parity updates during data migration to reduce the numbers of additional XOR computations and disk I/Os. Third, it outsources all parity updates brought by RAID scaling to a surrogate disk. Fourth, it delays synchronizing the modified layout metadata onto disks to minimize the number of metadata writes without compromising data reliability.

McPod satisfies all the six requirements shown in Section 1.2. We implement the McPod approach in the software RAID of Linux Kernel 2.6.32.9. The benchmark studies on the three real-system workloads (i.e., Financial1, TPC-C, and WebSearch2) show that McPod outperforms the existing "moving-everything" approach by 67.78-79.64 percent in redistribution time and by 14.24-27.16 percent in user response time. Our experiments also illustrate that the performance of the RAID scaled using McPod is almost identical to that of the round-robin RAID.

## 2 Related Work

Block-level RAID scaling approaches are divided into two categories: optimizing the process of data migration and reducing the amount of data to be moved.

### 2.1 Optimizing Data Migration for RAID Scaling

The conventional approaches to RAID scaling redistribute data and preserve the round-robin order after adding disks. Gonzalez and Cortes [12] proposed a gradual assimilation approach (GA) to control the speed of RAID-5 scaling. Brown designed a reshape toolkit in a Linux MD driver (MD-Reshape) [13]. It writes layout metadata with a fixed-sized data window. User requests to the data window have to queue up until all data blocks within the window are moved. Therefore, the window size cannot be too large. Metadata updates are quite frequent.

The MDM method [15] eliminates the parity modification cost of RAID-5 scaling by exchanging some data blocks between original disks and new disks. However, it does not guarantee an even data and parity distribution. Also, it does not increase (just maintains) the data storage efficiency after adding more disks.

A patent [16] presents a method to eliminate the need to rewrite the original data blocks and parity blocks on original disks. However, the obvious uneven distribution of parity blocks will bring a penalty to write performance.

Franklin and Wong [17] proposed to use spare disks to provide immediate access to new space. During data redistribution, new data are mapped to spare disks. Upon completion of the redistribution, new data are copied to the set of data disk drives. This kind of method requires spare disks to be available.

Zhang et al. [14], [18] discovered that there is always a reordering window during data redistribution for round-robin RAID scaling. By leveraging this insight, they proposed the ALV approach to improving the efficiency of RAID-5 scaling. However, ALV still suffers from large data migration.

### 2.2 Reducing Data Migration for RAID Scaling

With the development of object-based storage, randomized RAID [19], [20], [21] is now gaining the spotlight in the data placement area. Seo and Zimmermann [22] 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. RUSH [23], [24] and CRUSH [25] 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. The random slicing strategy [26] keeps a small table with information about previous insert and remove operations, significantly reducing the required amount of randomness while delivering a uniform load distribution. These randomized strategies are designed for object-based storage systems. They only provide mapping from logical addresses to a set of storage devices, while the data placement on a storage device is resolved by additional software running on the device itself.

Wu and He [27] proposed the GSR approach to accelerating RAID-5 scaling. GSR moves the second section of data onto the new disks, while keeping the first section of data unmoved. Only the original disks serve accesses to the first section of data. Only the new disks serve accesses to the second section of data. GSR minimizes

data migration and parity modification. The main limitation of GSR is the performance of RAID systems after scaling.

The FastScale approach to RAID-0 scaling [6], [7] minimizes data migration while maintaining a uniform data distribution. FastScale provides a good starting point for RAID-4 scaling. However, RAID-4 scaling is more challenging, as discussed in Section 1.2.

## 3 How McPod Performs RAID Scaling

McPod accelerates RAID-4 scaling with five techniques, i.e., minimizing data migration, coalesced data accesses, piggyback and parallel parity updates, outsourcing parity updates, and delayed metadata updates. The first two techniques improve the migration efficiency of regular data. The third and fourth techniques reduce the overhead of parity updates during RAID scaling. The last one minimizes the number of metadata updates without compromising data reliability.

### 3.1 Minimizing Data Migration

During RAID-4 scaling, McPod moves a fraction of existing data blocks from original disks to new disks. The goal is that data migration is minimal while data distribution across all the disks is uniform. RAID-4 scaling is over when data migration is finished. After that, newly added capacity is available, and new data can be filled into the RAID-4 gradually. This section focuses on an overview of how McPod minimizes data migration for RAID-4 scaling. For more details on data migration and data filling, see the description in Section 4.1.

To understand how the McPod approach works and how it minimizes data migration while maintaining uniform data distribution across all the disks, we take the $i$th RAID scaling operation from $N_{i-1}$ disks to $N_i$ as an example. We suppose each disk consists of $s$ data blocks. Before this scaling operation, there are $(N_{i-1}-1)\times s$ data blocks stored on $N_{i-1}-1$ data disks. The $s$ parity blocks on the parity disk will keep unmoved.

As shown in Fig. 1, each $N_i-1$ consecutive locations in a data disk are grouped into a *segment*. For the $N_i-1$ data disks, $N_i-1$ segments with the same physical address are grouped into one *region*. Locations on all disks with the same block number form a *column* or a *stripe*. In Fig. 1, different regions are separated by wavy lines. For different regions, the ways for data migration and data filling are completely identical. Therefore, we will focus on one region, and let $s_1 = N_i - 1$ be the number of data blocks in one segment.

In a region, all data blocks within a parallelogram will be moved. The base of the parallelogram is $N_i - N_{i-1}$, and the height is $N_{i-1}-1$. In other words, $N_i - N_{i-1}$ data blocks are selected from each old data disk and migrated to new disks. The $N_i - N_{i-1}$ blocks are consecutive, and the start address is the disk number *disk_no*. Fig. 1 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.

For RAID-4 scaling, it is desirable to ensure an even load distribution on all data disks and minimal block

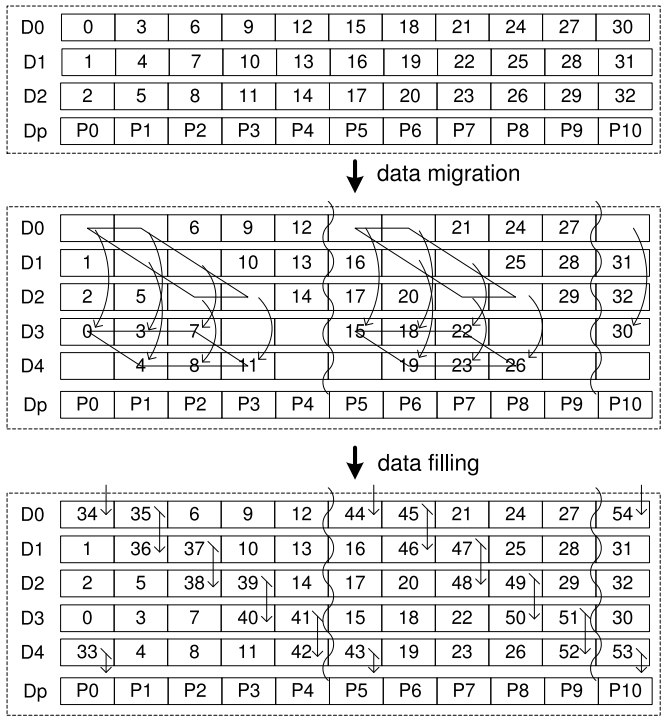



Fig. 1. RAID-4 scaling from four disks to six using McPod.

movement. After data migration, each data disk, either old or new, has $N_{i-1}-1$ data blocks. That is to say, McPod regains a uniform data distribution. The total number of data blocks to be moved is $(N_i - N_{i-1})\times(N_{i-1}-1)$. This reaches the minimal number of moved blocks in each region, i.e., $((N_{i-1}-1)\times s_1)\times(N_i - N_{i-1})/(N_i-1) = (N_{i-1}-1)\times(N_i - N_{i-1})$. We can claim that the RAID scaling using McPod minimizes data migration while maintaining uniform data distribution across all data disks.

After RAID-4 scaling, new data can be filled gradually. As shown in Fig. 1, $N_i - N_{i-1}$ new data blocks are placed into each stripe consecutively. These new blocks are distributed in a round-robin order.

### 3.2 Access Coalescing and Parallel I/O

McPod only moves 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 for large sequential accesses, McPod coalesces multiple accesses to physically successive blocks into a single I/O.

Take a RAID-4 scaling from four disks to six as an example, shown in Fig. 2. Let us focus on the first region. McPod 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 to read blocks 8 and 11, simultaneously. This means that to read all these blocks, McPod requires only three I/Os, instead of six.

When all the six blocks have been read into a memory buffer, McPod issues the first I/O request to write blocks 0, 3, and 7, and the second I/O request to write blocks 4, 8, and 11, simultaneously (see Fig. 3). In this way, only two

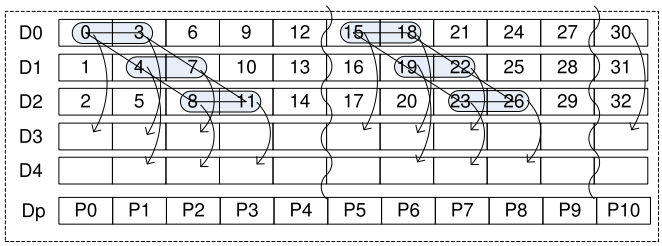



Fig. 2. Coalesced reads for RAID-4 scaling from four disks to six. Multiple successive blocks are read via a single I/O.

large sequential write requests are issued as opposed to six small writes.

For RAID-4 scaling from $N_{i-1}$ disks to $N_i$ disks, $N_{i-1}-1$ reads and $N_i - N_{i-1}$ writes are required to migrate all the data in a region, i.e., $(N_{i-1}-1) \times (N_i - N_{i-1})$ data blocks. Furthermore, all these large-size data reads (or writes) are on different disks. They can be done in parallel, further increasing I/O rate.

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 32 KB or 64 KB [13], [28], [29]. Thus, accessing multiple successive blocks via a single I/O enables McPod 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.

### 3.3 Piggyback and Parallel Parity Updates

RAID-4 arrays can tolerate one member drive failure by maintaining the parity information as the XOR sum of all the data blocks within a stripe. Copying some blocks within a stripe, as McPod does, changes the total contents of the stripe, and therefore requires a parity update. McPod piggybacks parity updates during data migration to minimize the numbers of additional XOR computations and disk I/Os.

To consider how one parity block is updated, we divide data stripes in the RAID into two categories. In a stripe of the first category, no data migration is required. As shown in Fig. 4, stripe $4$ in the first region is in this category. In this case, McPod does not change any content of the stripe, and therefore does not require a parity update. Therefore, no additional XOR computation and disk I/O is needed.

In a stripe of the second category, data migration is required. As shown in Fig. 4, stripes $0$, $1$, $2$, $3$ in the first region are in this category. Without loss of generality, we assume that blocks in a stripe are $B_0$, $B_1$, $B_2$, and $P$ before

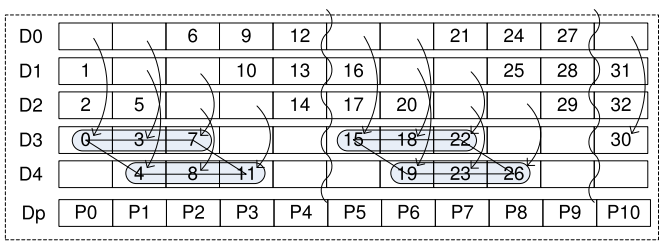



Fig. 3. Coalesced writes for RAID-4 scaling from four disks to six. Multiple successive blocks are written via a single I/O.

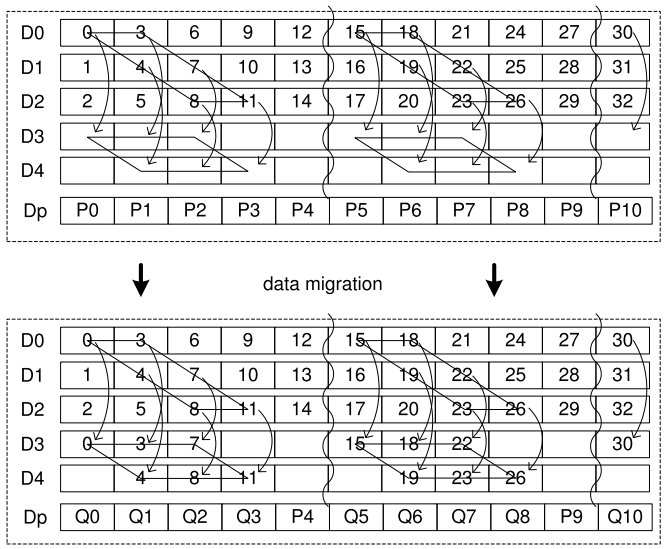



Fig. 4. Parity changes across data migration. Copying some blocks within a stripe changes the total contents of the stripe, and therefore requires a parity update.

scaling. After data migration, blocks in this stripe are $B_0$, $B_1$, $B_2$, $B_1$, $B_2$, and $P'$. Since $P = B_0 \oplus B_1 \oplus B_2$, we have $P' = B_0 \oplus B_1 \oplus B_2 \oplus B_1 \oplus B_2 = P \oplus B_1 \oplus B_2$. Since $B_1$ and $B_2$ need to be moved, they will be in the memory buffer and available for computing parity. As a result, to maintain a consistent parity, only a parity read and a parity write are added. In addition, all these three reads are on three disks, and therefore can be done in parallel. Similarly, the three writes can also be done in parallel. This parallelism will further reduce the cost of parity updates.

### 3.4 Outsourcing Parity Updates

Piggyback and parallel parity updates succeed in reducing the cost of maintaining a consistent parity during RAID-4 scaling. However, the parity disk is read and written frequently during scaling, which tends to become a performance bottleneck. McPod effectively outsources all parity updates brought by RAID scaling to a surrogate disk during RAID scaling.

In the process of RAID scaling, original parity blocks are read from the parity disk. The new parity blocks are calculated and written into the surrogate disk. Fig. 5 illustrates a state in the scaling process. In this state, parity accesses are

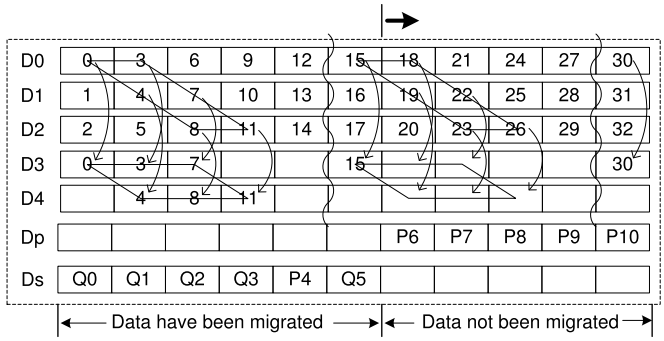



Fig. 5. All parity updates brought by RAID scaling are outsourced to a surrogate disk, $D_s$.

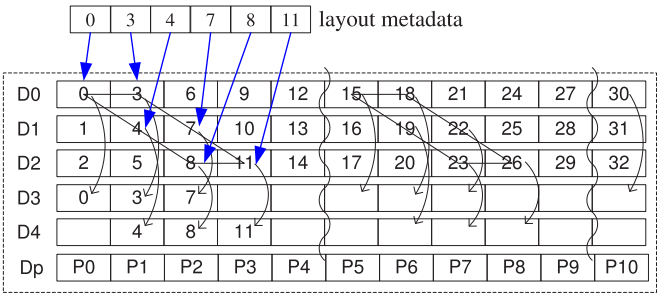



Fig. 6. If data blocks are copied to their new locations and metadata are not yet updated when the system fails, data consistency is still maintained, because the data in their original locations are valid and available.

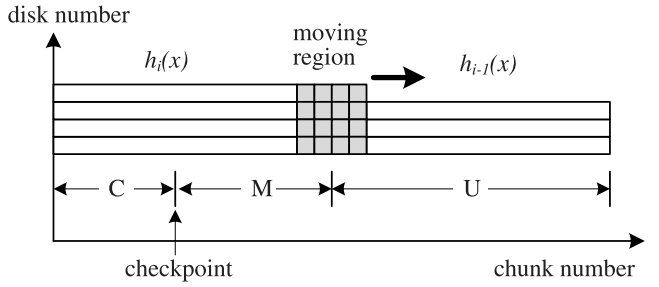



Fig. 7. Delayed updates of layout metadata. "C": migrated and checkpointed; "M": migrated but not checkpointed; "U": not migrated. Data redistribution is checkpointed only when a user write request arrives in the area "M".

redirected to the surrogate disk, $D_s$, if data in the target stripe have been migrated. Otherwise, parity accesses are performed on the parity disk, $D_p$. In this manner, the surrogate disk also offloads a proportion of parity accesses brought by user writes. As the scaling operation progresses, the surrogate disk holds more parity blocks. Accordingly, the surrogate disk offloads increasingly much I/O workload from the parity disk, thus reducing the response times of the I/O requests served by the RAID set under scaling. When the RAID scaling is finished, the surrogate disk will become the new parity disk, and the original parity disk will be released from this RAID set.

The device overhead introduced by outsourcing parity updates is only a surrogate disk. McPod is designed for use in a large storage system consisting of multiple RAID sets and a few hot spare disks. In such an environment, a hot spare disk can act as a surrogate disk during RAID scaling. After RAID scaling, the original parity disk will be released. The numbers of hot spare disks are identical across a scaling operation. Therefore, the device overhead of a surrogate disk in McPod can be ignored.

### 3.5 Delayed Metadata Updates

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 layout metadata do not get updated until all blocks have been moved, data consistency may be destroyed. Ordered operations [29] of copying a data block and updating the layout metadata (a.k.a., *checkpoint*) can ensure data consistency. However, ordered operations cause each block movement to require one metadata write, which results in a large cost of data migration. Because metadata are usually stored at the beginning of all member disks, each metadata update causes one long seek per disk. McPod delays metadata updates to minimize the number of metadata writes without compromising data consistency.

The foundation of delayed metadata updates 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 layout metadata have not been updated (see Fig. 6), 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 they were 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 delayed metadata updates is that data blocks are copied to new locations continuously, while the modified layout metadata are not synchronized onto disks until a threat to data consistency appears. We use $h_i(x)$ to describe the geometry after the $i$th scaling operation, where $N_i - 1$ data disks serve user requests. Fig. 7 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 layout metadata synchronized onto disks.

Since one write of metadata can store multiple layout changes of data blocks, delayed updates can significantly decrease the number of metadata updates, reducing the cost of data migration. Furthermore, delayed metadata updates 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 How McPod Addresses Data

In a RAID system, data addressing includes how to map a logical address to its physical address and how to map a physical address to its logical address. They are termed a mapping algorithm and a demapping algorithm respectively.

### 4.1 The Mapping Algorithm

Fig. 8 shows the mapping algorithm to minimize data migration required by RAID scaling. An array $N$ is used to record the history of RAID scaling. $N[0]$ is the initial number of disks in the RAID. After the $i$th scaling operation, the RAID consists of $N[i]$ disks.

**Algorithm**: Mapping($t, N, s, x, d, b$).

**Input**: The input parameters are $t$, $N$, $s$, and $x$, where
$t$: the number of scaling times;
$N$: the scaling history ($N[0], N[1], ..., N[t]$);
$s$: the number of data blocks in one disk;
$x$: a logical block number.
**Output**: The output data are $d$ and $b$, where
$d$: the disk holding block $x$;
$b$: the physical block number on disk $d$.

```
(1)  if (t = 0) then
(2)      m ← N[0]; //the number of initial disks
(3)      d ← x mod (m − 1);
(4)      b ← x/(m − 1);
(5)      return;
(6)  end if;
(7)  m ← N[t − 1]; //the number of old disks
(8)  n ← N[t] − m; //the number of new disks
(9)  if (0 ≤ x ≤ (m − 1) × s − 1) then //an old data block
(10)     Mapping(t − 1, N, s, x, d_0, b_0);
(11)     b_1 ← b_0 mod (m + n − 1);
(12)     if (d_0 ≤ b_1 ≤ d_0 + n − 1) then //to be moved
(13)         d ← Moving(d_0, b_1, m, n);
(14)         b ← b_0;
(15)     else //not to be moved
(16)         d ← d_0;
(17)         b ← b_0;
(18)     end if;
(19) else //a new data block
(20)     Placing(x, m, n, s, d, b);
(21) end if.
```

Fig. 8. The Mapping algorithm used in McPod.

When a RAID is constructed from scratch (i.e., $t = 0$), it is actually a round-robin RAID. The address of block $x$ can be calculated via one division and one modular operations (lines 3-4).

Let us examine the $t$th scaling, where $n$ disks are added into a RAID made up of $m$ disks (lines 7-8). For an original block (line 9), McPod calculates its old address ($d_0$, $b_0$) before the $t$th scaling (line 10). If the block ($d_0$, $b_0$) needs to be moved during the $t$th scaling (line 12), McPod changes the disk ordinal number via the Moving() function (line 13) while keeps the block ordinal number unchanged (line 14). For a new block (line 19), McPod places it via the Placing() procedure (line 20).

The code of line 12 is used to decide whether a data block ($d_0$, $b_0$) will be moved during a RAID scaling. As shown in Fig. 1, there is a parallelogram in each region. The base of the parallelogram is $n$, and the height is $m-1$. 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 ending columns 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.

The Moving() function is depicted as follows. As shown in Fig. 9, a migrating parallelogram is divided into three parts: a head triangle, a body parallelogram, and a tail triangle. How a data block moves depends on which part it lies in. The head triangle and the tail triangle keep their shapes unchanged. The head triangle will be moved by $m-1$ disks, while the tail triangle will be moved by $n$ disks. However, the body is sensitive to the relationship between $m-1$ and $n$. The body is twisted from a parallelogram to a rectangle when $m-1 \geq n$, and from a rectangle to a parallelogram when $m-1 < n$. McPod keeps the relative locations of all data blocks in the same column.

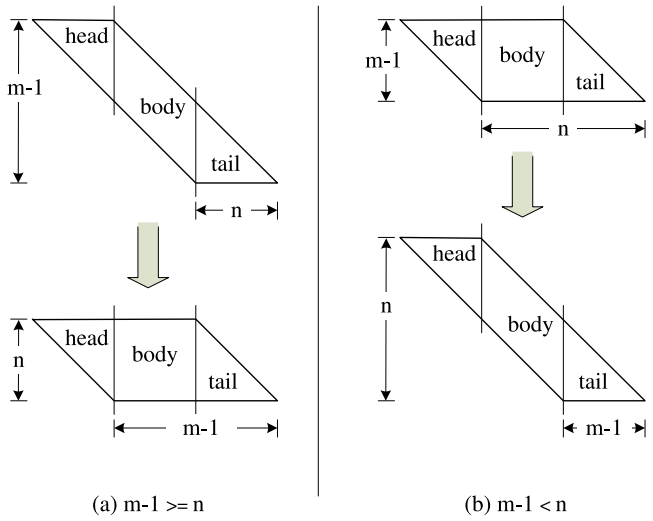



Fig. 9. The variation of data layout involved in migration.

The Placing() procedure is used to address a block $x$ that is newly added after the last scaling. Each stripe holds $n$ new blocks. Suppose block $x$ is the $y$th new block. We have $b = y/n$. The order of placing new blocks in each stripe is shown in Fig. 1.

### 4.2 The Demapping Algorithm

In many cases, it is also required to map a physical address to a logical address. McPod provides such a mapping with the *Demapping* algorithm, shown in Fig. 10.

Let us examine the $t$th scaling, where $n$ disks are added into a RAID made up of $m$ disks (lines 6-7). The logical address of block $(d, b)$ can be calculated in a recursive manner.

- If block $(d, b)$ is an original block and is not moved (line 9), the logical address of block $(d, b)$ keeps unchanged across the $t$th scaling (line 10).
- If block $(d, b)$ is an original block and is moved (line 12), McPod gets its original location $(d_0, b)$ before the $t$th scaling via the Demoving() function (line 13). It should be remembered that McPod changes the disk ordinal number while keeping the block ordinal number unchanged. Then, McPod calculates the logical address of block $(d_0, b)$ before the $t$th scaling (line 14).
- If block $(d, b)$ is a new block, McPod gets its logical address via the *Deplacing()* function (line 16).

The code of line 9 is used to decide whether a data block $(d, b)$ is an old block and is not moved during this scaling. If and only if the data block is within a source parallelogram

**Algorithm**: Demapping$(t, N, s, d, b)$.

**Input**: The input parameters are $t$, $N$, $s$, $d$ and $b$, where
$t$: the number of scaling times;
$N$: the scaling history $(N[0], N[1], ..., N[t])$;
$s$: the number of data blocks in one disk;
$d$: the disk holding block $x$;
$b$: the physical block number on disk $d$.
**Output**: The output datum is data block $x$ at location $b$ of disk $d$.

```
if (t = 0) then                                                    (1)
    m ← N[0]; //the number of initial disks                        (2)
    x ← b × (m − 1) + d;                                           (3)
    return x;                                                      (4)
end if;                                                            (5)
m ← N[t − 1]; //the number of old disks                            (6)
n ← N[t] − m; //the number of new disks                            (7)
b1 ← b mod (m + n − 1);                                            (8)
if (0 ≤ d < m − 1) and (b1 < d or b1 > d + n − 1) then             (9)
    //an old and unmoved data block
    return Demapping(t − 1, N, s, d, b);                           (10)
end if;                                                            (11)
if (d ≥ m − 1) and (d − m + 1 ≤ b1 ≤ d − 1) then                   (12)
    //an old and moved data block
    d0 ← Demoving(d, b1, m, n);                                    (13)
    return Demapping(t − 1, N, s, d0, b);                          (14)
end if;                                                            (15)
return Deplacing(m, n, s, d, b). //a new data block                (16)
```

Fig. 10. The Demapping algorithm used in McPod.

(see Fig. 1), it is moved. Likewise, the code of line 12 is used to determine whether a data block $(d, b)$ is an old block and has been moved during this scaling. If and only if the data block is within a destination parallelogram, it has been moved during the $t$th scaling.

## 5 Performance Evaluation

This section mainly presents results of a comprehensive experimental evaluation comparing McPod with the existing "moving-everything" solution. This performance study analyzes their performance in terms of user response time and redistribution time.

### 5.1 Prototype and Evaluation Methodology

We implement McPod in the MD driver shipped with Linux Kernel 2.6.32.9. MD is a software RAID system, which uses MD-Reshape to scale RAID-4 volumes [13]. According to the addressing algorithm, MD forwards incoming I/O requests to corresponding disks. When RAID-4 scaling begins, MD creates a kernel thread to perform data redistribution. McPod cannot redistribute a new region until all the I/O requests already issued to this region are completed.

We evaluate our design by running trace-driven experiments over a real system. To replay I/O traces, we implement a block-level replay tool using Linux 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.

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

- Financial1 is from the storage performance council (SPC). It was collected from OLTP applications running at a large financial institution [30]. The write ratio is high.
- TPC-C traced disk accesses of the TPC-C database benchmark with 20 warehouses [31]. It was collected with one client running 20 iterations.
- WebSearch2 is also from SPC. It was collected from a system running a web search engine. The read-dominated WebSearch2 trace exhibits the strong locality in its access pattern.

The testbed used in these experiments is described as follows. Linux kernel 2.6.32.9 is installed on a machine with Intel Xeon 5606 2.13 GHz quad-core processor and 8 GB of memory. The file system used is EXT4. via a 6 GB/s SATA expansion card, 12 Seagate ST500DM002 SATA disks are connected to this machine.

### 5.2 The Scaling Efficiency

Each experiment lasts from the beginning to the end of data redistribution for RAID scaling. We focus on comparing redistribution times and user I/O latencies when different scaling programs are running in background.

The purpose of our first experiment is to quantitatively characterize the advantages of McPod through a comparison with MD-Reshape. We conduct a scaling operation of adding one disk to a four-disk RAID. To perform experiments in an acceptable time, we used disk partitions with a capacity of 100 GB, instead of the whole disks, to construct disk arrays. Each approach performs with the 64 KB chunk size under a Financial1 workload.

A group of rate-control parameters means a tradeoff between the redistribution time objective and the response time objective. Furthermore, unless both redistribution time and user response time using 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 ease of comparison, we choose control parameters for the different experiments. The parameters of "sync_speed_max" and "sync_speed_min" in MD-Reshape are set 200,000 and 2,000 respectively. In McPod, they are set 200,000 and 8,000 respectively. 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 application 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 1,000 seconds. Furthermore, we get a local average latency from each section. A local average latency is the average of I/O latency in a section. Fig. 11 plots local average latencies using the two approaches as the time increases along the $x$-axis. It illustrates that McPod demonstrates a noticeable improvement over MD-Reshape in two metrics. First, the redistribution time using McPod is significantly shorter than that using MD-Reshape. They

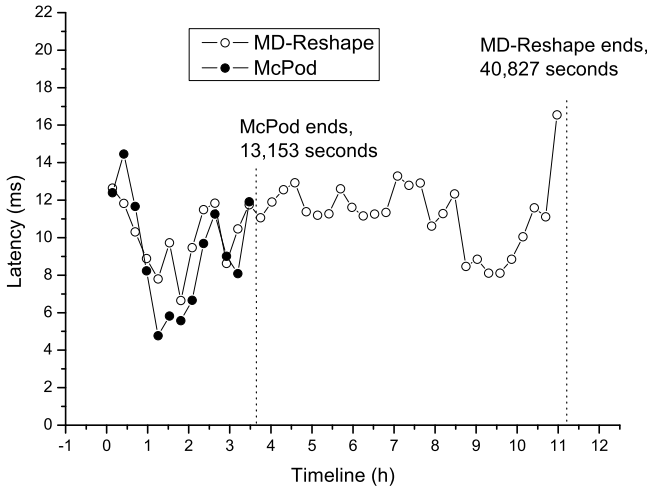



Fig. 11. Performance comparison between McPod and MD-Reshape for the Financial1 workload.

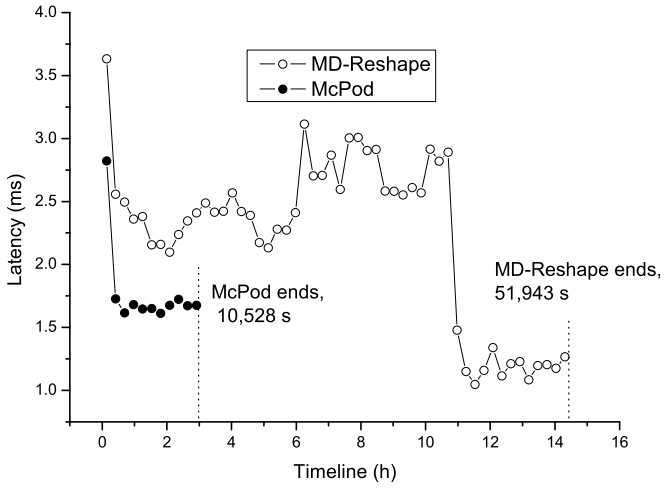



Fig. 13. Performance comparison between McPod and MD-Reshape for the TPC-C workload.

are 13,153 and 40,827 seconds, respectively. In other words, McPod has a 67.78 percent shorter redistribution time than MD-Reshape.

The main factor in McPod's reducing the redistribution time is the significant decline of the amount of the data to be moved. When MD-Reshape is used, 100 percent of data blocks have to be migrated. However, when McPod is used, only 25 percent of data blocks need to be migrated. Another factor is the effective exploitation of the other four optimization techniques. Coalesced data accesses improve the migration efficiency of regular data. Piggyback parity updates and outsourcing parity updates reduce the overhead of parity updates during RAID scaling. Delayed metadata updates minimizes the number of metadata updates without compromising data reliability.

Second, local average latencies of MD-Reshape are longer than those of McPod. The global average latency using MD-Reshape reaches 9.76 ms while that using McPod is 8.37 ms. In other words, McPod brings an improvement of 14.24 percent in user response time. Fig. 12 shows the cumulative distribution (CD) of user response time during data redistribution. To provide a fair comparison, I/Os involved in statistics for MD-Reshape are only those issued before 13,153 seconds. For any I/O latency smaller than 95.34 ms, the CD value of McPod is greater than that of MD-Reshape noticeably and consistently. This indicates again that McPod has smaller response time of user I/Os than MD-Reshape.

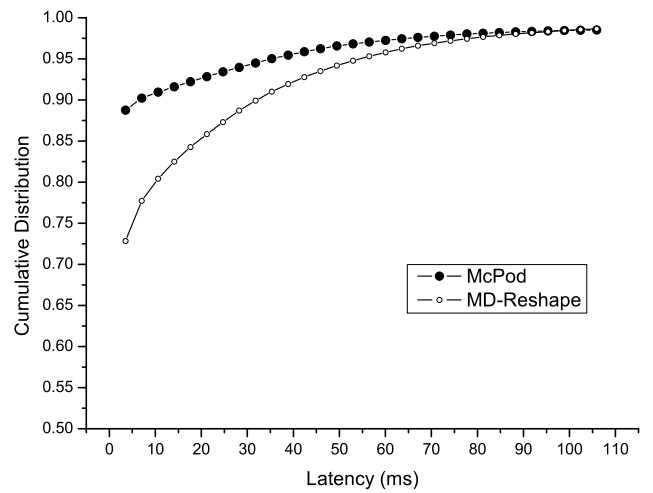



Fig. 12. Cumulative distribution of I/O latency during data redistribution by the two approaches for the Financial1 workload.

The reason for the improvement in user response time is explained as follows. During RAID scaling, data redistribution and foreground applications share and even contend for I/O resources in the system. McPod decreases the amount of the data to be moved significantly. Moreover, McPod minimizes metadata writes via delayed metadata updates. As a result, the RAID system has more time to serve applications. It is also noteworthy that due to significantly shorter data redistribution time, McPod has a markedly lighter impact on the user I/O latencies than MD-Reshape does.

A factor that might affect the benefits of McPod is the type of workload under which data redistribution performs. Under the TPC-C workload, we also measure the performance of McPod and MD-Reshape in performing the "$4+1$" scaling operation. The parameters of "sync_speed_max" and "sync_speed_min" in MD-Reshape are set 200,000 and 2,000 respectively. In McPod, they are set 200,000 and 10,000 respectively.

For the TPC-C workload, Fig. 13 shows local average latency versus the redistribution time for MD-Reshape and McPod. It shows once again the efficiency of McPod in improving the redistribution time. The redistribution times using MD-Reshape and McPod are 14.43 hours and 2.92 hours, respectively. That is to say, McPod has an improvement of 79.64 percent in the redistribution time. Likewise, local average latencies of McPod are also obviously shorter than those of MD-Reshape. The global average latency using McPod is 1.77 ms while that using MD-Reshape reaches 2.43 ms. In other words, McPod has an improvement of 27.16 percent in user response time.

We can draw one conclusion from the above two experiments. Under various workloads, McPod can consistently outperform MD-Reshape by 67.78-79.64 percent

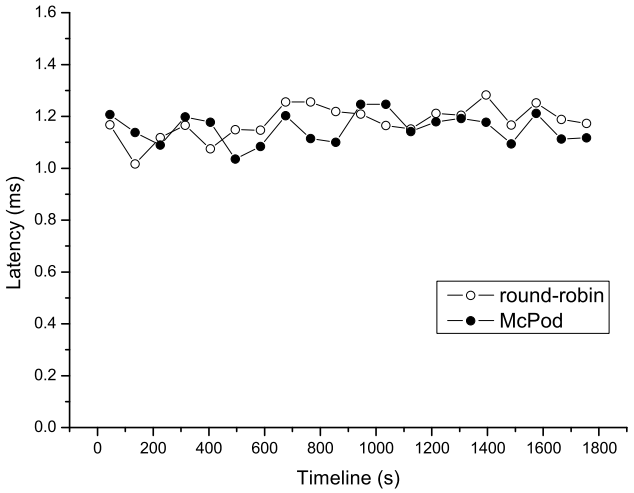



Fig. 14. Performance comparison between McPod layout and round-robin layout for the WebSearch2 workload after one scaling operation "4 + 1."

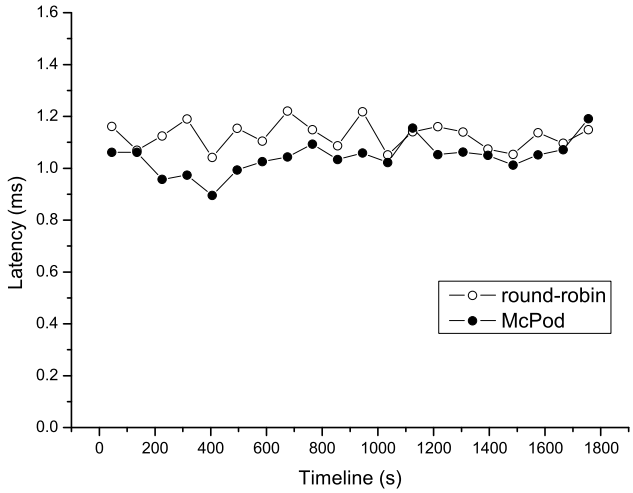



Fig. 16. Performance comparison between McPod layout and round-robin layout for the WebSearch2 workload after two scaling operations "4 + 1 + 1."

in redistribution time and by 14.24-27.16 percent in user response time.

### 5.3 The Performance After Scaling

The above experiments show that McPod improves the scaling efficiency of RAID-4 significantly. One of our concerns is whether there is a penalty in the performance of the data layout after scaling using McPod, compared with the round-robin layout preserved by MD-Reshape.

We use the WebSearch2 workload to measure the performance of the two RAIDs, scaled from the same RAID using McPod and MD-Reshape. Each experiment lasts 30 minutes, 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 performance of two RAIDs, after one scaling operation "4 + 1" using the two scaling approaches. Fig. 14 plots the local average latency of the two RAIDs as the time increases along the $x$-axis. We can find that the performance of the McPod RAID is almost identical to that of the round-robin RAID. With regard to the round-robin RAID, the average latency is 1.18 ms. For the McPod RAID, the average latency is 1.16 ms. Under the Financial1 and TPC-C workloads, we also measure the performance of the two RAIDs. Fig. 15 compares the average latencies of the two RAIDs. It illustrates that their performances are almost identical under each workload.

Second, we compare the performance of two RAIDs, after two scaling operations "4 + 1 + 1" using the two approaches. Fig. 16 plots local average latencies of the two RAIDs as the time increases along the $x$-axis. It reveals the difference in the performance of the two RAIDs. With regard to the round-robin RAID, the average latency is 1.13 ms. For the McPod RAID, the average latency is 1.05 ms. Fig. 17 compares the average latencies of the two RAIDs under different workloads. It again shows that their performances are almost identical under each workload.

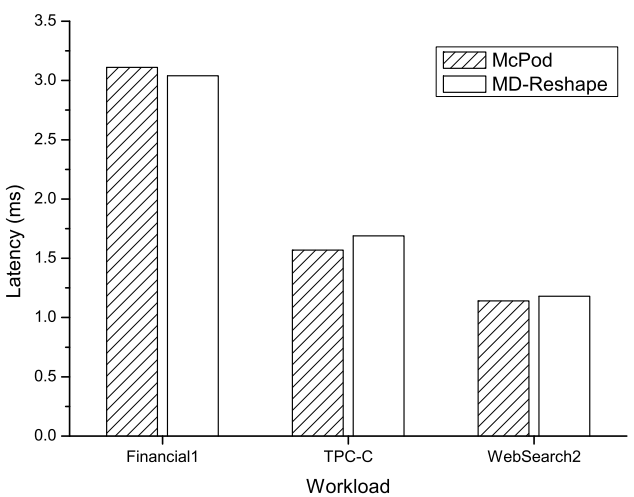



Fig. 15. Performance comparison between McPod layout and round-robin layout for different workloads after one scaling operation "4 + 1."

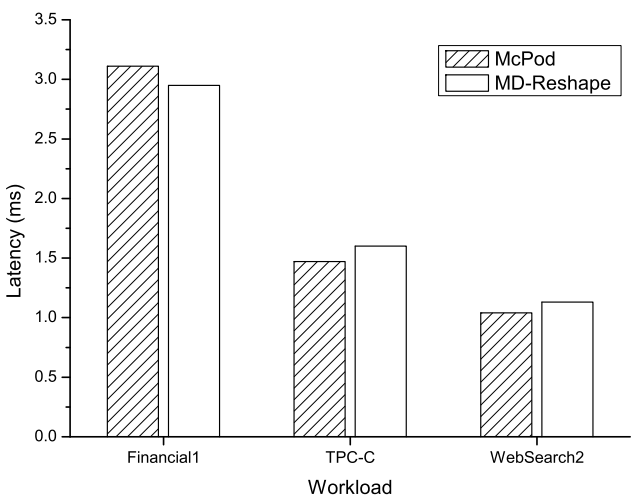



Fig. 17. Performance comparison between McPod layout and round-robin layout for different workloads after two scaling operations "4 + 1 + 1."

We can reach the conclusion that the performance of the RAID scaled using McPod is almost identical to that of the round-robin RAID scaled using MD-Reshape.

## 6 Conclusions

In order to regain load balance during RAID-4 scaling, it is necessary to redistribute data across all the disks. This paper presents McPod—a new data redistribution approach to accelerating RAID-4 scaling. First, with a new and elastic addressing function, McPod minimizes the number of data blocks to be migrated without compromising the uniformity of data distribution. Second, McPod optimizes online data migration with four unique techniques, i.e., coalesced accesses and parallel I/O, piggyback and parallel parity updates, outsourcing parity updates, and delayed metadata updates.

Our results from detailed experiments using real-system workloads show that, compared with MD-Reshape, a scaling toolkit released in 2010, McPod can reduce redistribution time by up to 67.78-79.64 percent and reduce user response time by 14.24-27.16 percent. The experiments also illustrate that the performance of the RAID scaled using McPod is almost identical to that of the round-robin RAID.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful comments and detailed suggestions, which have substantially improved the content and presentation of this paper. They also thank Lili Song for helpful comments and assistance with our experimentation. This work was supported by the National Natural Science Foundation of China under Grants 60903183, 61170008, and 61272055, and the National Grand Fundamental Research 973 Program of China under Grant No. 2014CB340402.

## References

[1] D.A. Patterson, G.A. Gibson, and R.H. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 109-116, 1988.

[2] P. Chen et al., "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145-185, June 1994.

[3] X. Yu et al., "Trading Capacity for Performance in a Disk Array," *Proc. Fourth Conf. Symp. Operating System Design and Implementation (OSDI '00)*, Oct. 2000.

[4] S. Ghandeharizadeh and D. Kim, "On-Line Reorganization of Data in Scalable Continuous Media Servers," *Proc. Seventh Int'l Conf. Database and Expert Systems Applications*, Sept. 1996.

[5] D.A. Patterson, "A Simple Way to Estimate the Cost of Down-Time," *Proc. 16th USENIX Conf. System Administration (LISA '02)*, pp. 185-188, Oct. 2002.

[6] W. Zheng and G. Zhang, "FastScale: Accelerate RAID Scaling by Minimizing Data Migration," *Proc. Ninth USENIX Conf. File and Storage Technologies (FAST '11)*, Feb. 2011.

[7] G. Zhang, W. Zheng, and K. Li, "Design and Evaluation of a New Approach to RAID-0 Scaling," *ACM Trans. Storage*, vol. 9, no. 4, article 11, Nov. 2013.

[8] G. Zhang, W. Zheng, and K. Li, "Rethinking RAID-5 Data Layout for Better Scalability," *IEEE Trans. Computers*, to be published, 2014.

[9] Y. Mao, J. Wan, Y. Zhu, and C. Xie, "A New Parity-Based Migration Method to Expand RAID-5," *IEEE Trans. Parallel and Distributed Systems*, to be published, 2014.

[10] G. Zhang, K. Li, J. Wang, and W. Zheng, "Accelerate RDP RAID-6 Scaling by Reducing Disk I/Os and XOR Operations," *IEEE Trans. Computers*, to be published, 2014.

[11] O. Rodeh, "The Write-Anywhere-File-Layout (WAFL)," http://www.cs.tau.ac.il/~ohadrode/slides/WAFL.pdf, 2014.

[12] 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.

[13] N. Brown, Online RAID-5 Resizing. drivers/md/raid5.c in the source code of Linux Kernel 2.6.32.9. http://www.kernel.org/, Feb 2010.

[14] G. Zhang, W. Zheng, and J. Shu, "ALV: A New Data Redistribution Approach to RAID-5 Scaling," *IEEE Trans. Computers*, vol. 59, no. 3, pp. 345-357, Mar. 2010.

[15] S.R. Hetzler, *Data Storage Array Scaling Method and System with Minimal Data Movement*, US Patent 20,080,276,057, 2008.

[16] "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 6,000,010, 1999.

[17] 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.

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

[19] A. Goel, C. Shahabi, S-Y.D Yao, and R. Zimmermann, "SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks," *Proc. 18th Int'l Conf. Data Engineering (ICDE '02)*, pp. 473-482, 2002.

[20] J.R. Santos, R.R. Muntz, and B.A. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems*, pp. 44-55, 2000.

[21] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, Distributed Data Placement Strategies for Storage Area Networks (Extended Abstract)," *Proc. ACM 12th Ann. Symp. Parallel Algorithms and Architectures*, pp. 119-128, 2000.

[22] B. Seo and R. Zimmermann, "Efficient Disk Replacement and Data Migration Algorithms for Large Disk Subsystems," *ACM Trans. Storage*, vol. 1, no. 3, pp. 316-345, Aug. 2005.

[23] R.J. Honicky and E.L. Miller, "A Fast Algorithm for Online Placement and Reorganization of Replicated Data," *Proc. 17th Int'l Parallel and Distributed Processing Symp. (IPDPS '03)*, Apr. 2003.

[24] R.J. Honicky and E.L. Miller, "Replication under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution," *Proc. 18th Int'l Parallel and Distributed Processing Symp. (IPDPS '04)*, 2004.

[25] S.A. Weil, S.A. Brandt, E.L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," *Proc. ACM/IEEE Conf. Super Computing (SC '06)*, 2006.

[26] A. Miranda, S. Effert, Y. Kang, E.L. Miller, A. Brinkmann, and T. Cortes, "Reliable and Randomized Data Distribution Strategies for Large Scale Storage Systems," *Proc. 18th Int'l Conf. High Performance Computing (HiPC '11)*, Dec. 2011.

[27] C. Wu and X. He, "GSR: A Global Stripe-Based Redistribution Approach to Accelerate RAID-5 Scaling," *Proc. 41st Int'l Conf. Parallel Processing (ICPP)*, pp. 460-469, 2012.

[28] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. third ed., Morgan Kaufmann Publishers, 2003.

[29] 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.

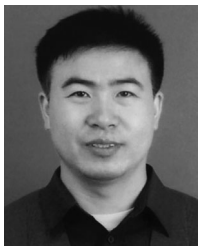
[30] OLTP Application I/O and Search Engine I/O, UMass Trace Repository, http://traces.cs.umass.edu/index.php/Storage/Storage, 2014.

[31] Performance Evaluation Laboratory, Brigham Young University, *Trace Distribution Center*, http://tds.cs.byu.edu/tds/, 2002.

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

**Jigang Wang** completed postdoctoral research at Tsinghua University in 2009. He is currently a senior fellow in the Strategy Planning Department at ZTE corporation. His research interests include cluster computing, network storage, and mobile communication.

**Keqin Li** received the BS degree in computer science from Tsinghua University in 1985, and the PhD degree in computer science from the University of Houston in 1990. He is a SUNY distinguished professor of computer science in the State University of New York at New Paltz. His research interests include design and analysis of algorithms, parallel and distributed computing, and computer networking.



**Jiwu Shu** received the PhD degree in computer science from Nanjing University in 1998, and finished the postdoctoral position research at Tsinghua University in 2000. Since then, he has been teaching at Tsinghua University. His current research interests include storage security and reliability, non-volatile memory based storage systems, and parallel and distributed computing. He is a member of the IEEE.

**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.