Xscale: Online X-Code RAID-6 Scaling Using Lightweight Data Reorganization

Guangyan Zhang, Guiyong Wu, Yu Lu, Jie Wu, and Weimin Zheng

Abstract—Disk additions to a RAID-6 storage system can simultaneously increase the I/O parallelism and expand the storage capacity. To regain a balanced load among both old and new disks, RAID-6 scaling requires moving certain data blocks onto newly added disks. Existing approaches to RAID-6 scaling are restricted by preserving a round-robin data distribution, and require migrating all the data, resulting in an expensive cost for RAID-6 scaling. In this paper, we propose Xscale, a new approach to accelerating X-code RAID-6 scaling by using lightweight data reorganization. Xscale minimizes the number of data blocks that require being moved, while maintaining a uniform data distribution across all disks. Furthermore, Xscale eliminates metadata updates while guaranteeing data consistency and data reliability. Compared with the round-robin approach, Xscale reduces the number of blocks to be moved by 63.6-89.5 percent, decreases the reorganization time by 35.62-37.26 percent, and reduces the I/O latency by 23.29-37.74 percent while the scaling programs are running in the background. In addition, there is no penalty in the performance of the data layout after scaling using Xscale, compared with the layouts maintained by other existing scaling approaches.

Index Terms—RAID-6 scaling, load balance, data migration, data reorganization, metadata update

1 Introduction

RAID-6 [1], [2], [3] storage systems provide a large I/O bandwidth via parallel I/O operations, and tolerate two disk failures by maintaining dual parity. RAID-6 has received more attention, since the probability of multiple disk failures is higher than ever [4], [5]. RAID-based architectures are also used in clusters and large-scale storage systems [6], [7]. Due to the ever-increasing demand of storage capabilities, applications often require larger storage capacities and higher performance. This is normally achieved by adding new disks to the existing RAID-6 volume [8], [9]. This disk addition is termed as *RAID-6 scaling*.

Several challenges arise when performing RAID-6 scaling. First, in order to regain uniformity in the data distribution, certain blocks must be moved to the new disks. Second, RAID-6 scaling has additional overhead of recalculating and updating the associated parities, as well as the necessary *metadata updates* to checkpoint the progress of data reorganization. Third, RAID solutions are widely used in online services where clients and applications need constant access to data. With these services, the downtime cost can be extremely high [10], and thus any strategy to scale RAID arrays should be able to interleave its job with normal I/O operations. Therefore, RAID-6 scaling requires an efficient approach to performing online data reorganization.

There are multiple coding methods proposed for RAID-6 arrays. According to the layout of data and parity, RAID-6 codes are categorized into horizontal codes [3], [11] and vertical codes [12], [13], [14]. Existing scaling approaches are proposed for the general case in RAID-0 or RAID-5 [15], [16], [17], [18], [19], [20], [21]. They cannot adapt to various coding methods in RAID-6, and therefore are not suitable for RAID-6 scaling. An efficient approach to RAID-6 scaling should be designed based on the characteristics of each coding method, respectively. The X-code [14] is an elegant erasure code for two-parity systems that, encodes, decodes and updates optimally. This paper focuses on the problem of X-code RAID-6 scaling.

Typical RAID scaling approaches [16], [18], [19], [20] preserve the round-robin data distribution after adding disks. However, they end up redistributing all the data between old and new disks, regardless of the numbers of new and old disks. Moving large amounts of data means either data reorganization will be completed in a long time, or the impact of data reorganization on application performance will be significant. There are some optimizations of data migration [16], [20] proposed for RAID scaling, e.g., I/O aggregation and rate control. They can improve the performance of RAID-6 scaling to a certain extent, but still suffer from large data migration and frequent metadata updates.

1.1 How X-Code Works

X-code is a vertical array code of size $N \times N$, and N is the number of disks in the RAID, and it should be a prime number greater than 2. X-code has a simple geometry, and each stripe contains $N-2$ data rows and 2 parity rows. Fig. 1 shows an example of X-code's data layout. Each data block in a coded array is protected by an anti-diagonal parity and a diagonal parity. The parity blocks are calculated by *XOR* operations. An example of anti-diagonal parity is calculated

- *G. Zhang, G. Wu, Y. Lu, and W. Zheng are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: {gyzh, zwm-dcs}@tsinghua.edu.cn, wugy10@gmail.com, luyu13@mails.tsinghua.edu.cn.*
- *J. Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, PA 19122. E-mail: jiewu@temple.edu.*

Manuscript received 19 Aug. 2015; revised 5 Feb. 2016; accepted 29 Feb. 2016. Date of publication 16 Mar. 2016; date of current version 16 Nov. 2016. Recommended for acceptance by J. L. Träff.
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.2016.2542806

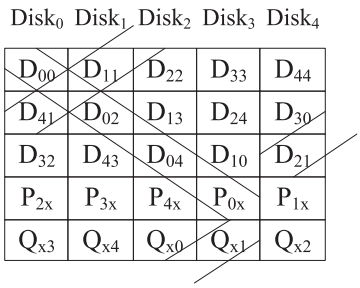


Fig. 1. The data layout of an X-code array with 5 disks.

by $P_{0x} = D_{00} \oplus D_{02} \oplus D_{04}$; an example of diagonal parity is calculated by $Q_{x1} = D_{21} \oplus D_{41} \oplus D_{11}$.

For an X-code array, distributions of data and parities are naturally uniform among all disks, so overheads of disk I/Os can be evenly shared by each disk. However, X-code's layout also leads to tight coupling among the disks in the RAID, so X-code has poor scalability, which needs improvement.

1.2 Initial Idea and Technical Challenges

Our initial idea for improving the efficiency of X-code RAID-6 scaling is that of minimizing data migration in the process of RAID-6 scaling. Zheng and Zhang [21] proposed the FastScale approach to accelerate RAID-0 scaling by minimizing data migration and lazy metadata updates. FastScale provides a good starting point for the efficient scaling of RAID-6 arrays. However, optimizing data reorganization for RAID-6 scaling will be more difficult, due to the need to maintain consistent, dual parities.

First, X-code has a uniform layout of data and parities, and naturally achieves load balancing in the RAID, so the uniformity should be maintained after scaling. Therefore, one of the challenges is to design a new data reorganization scheme for X-code scaling, to regain a uniform data distribution after scaling. Second, the overheads of data reorganization should be reduced, to accelerate RAID scaling and avoid significant degradation of access performance. Therefore, another technical challenge is to minimize data migration in the process of data reorganization. Third, using lazy metadata updates in X-code scaling will result in data loss when the system crashes and one disk fails. Therefore, the last technical challenge is to reduce the number of metadata updates, while guaranteeing data consistency and data reliability.

1.3 Design Objectives

To regain load balancing after adding disks to a storage system, RAID-6 scaling requires moving data blocks from old disks to new disks, and it is desirable to minimize the data migration. After scaling, a block may be moved to a new location, so we should provide a fast addressing algorithm for mapping a logical address to the corresponding physical address with a low complexity.

Assume that each disk consists of s blocks, and that there are N_{i-1} old disks and $N_i - N_{i-1}$ new disks during the ith RAID-6 scaling. Before this scaling, there are d_{i-1} data blocks and p_{i-1} parity blocks on each old disk. RAID-6 scaling should satisfy the following five requirements, while reorganizing the data.

- *Uniform data distributions.* After the ith RAID-6 scaling, each one of the N_i disks holds $(d_{i-1} \times N_{i-1})/N_i$ data blocks.
- *Minimal data migration.* During RAID-6 scaling, no data block should be exchanged among old disks, and the expected number of data blocks to be moved is $(N_i - N_{i-1}) \times ((d_{i-1} \times N_{i-1})/N_i)$.
- *Fast data addressing.* We should provide an addressing algorithm for calculating the new location of a block after scaling, and the addressing process should have low time and space complexities, to avoid a great impact on application performance.
- *Data consistency across system crash.* Data consistency should be guaranteed, if the system crashes during a scaling operation.
- *Data reliability during scaling.* Data reliability should be guaranteed, if the system crashes and/or if any two disks fail during a scaling operation.

To meet the above requirements for scaling, we propose *Xscale*, a new approach to accelerating X-code RAID-6 scaling by using lightweight data reorganization.

1.4 Paper Organization

The rest of the paper is organized as follows. In Section 2, we present how Xscale accelerates RAID-6 scaling. In Section 3, we describe our addressing algorithm in detail. In Section 4, we prove the properties of Xscale in theory. In Section 5, we quantitatively characterize the properties of Xscale through extensive simulation experiments. In Section 6, we evaluate Xscale by comparing it with existing approaches. In Section 7, we review some related work. Finally, we conclude this paper in Section 8.

2 Xscale Overview

Xscale accelerates online X-code RAID-6 scaling using lightweight data reorganization. The goal of lightweight data reorganization is achieved with two techniques, i.e., minimizing data migration and eliminating metadata updates.

2.1 Minimizing Data Migration

During RAID-6 scaling, Xscale moves data blocks from old disks to new disks, to balance load across all the disks, while Xscale minimizes the amount of data migration. After data migration, RAID-6 scaling is over, and new data can be placed into the disks. This section describes the rules of how Xscale migrates data from old disks to new disks.

2.1.1 View Transformation by Splitting Data and Parity

Without loss of generality, we focus on the ith scaling operation from N_{i-1} to N_i disks as an example, to describe how Xscale minimizes data migration. Each disk consists of s blocks, and there are d_{i-1} data blocks and p_{i-1} parity blocks on each old disk before scaling.

Suppose that the *physical view* describes the actual data layout on disks, and by the splitting operation, we convert it into the *logical view*, where we consider how to carry out data migration. Locations on all disks with the same block number form a *row*.

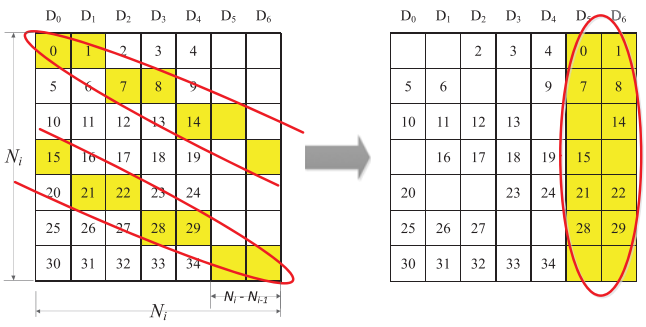


Fig. 2. Xscale's data migration during RAID-6 scaling from 5 to 7 disks.

In the logical view, data blocks are separated from parity blocks. All the data rows are pieced together into a *data segment* of d_{i-1} rows, and all the parity rows make up a *parity segment* of p_{i-1} rows. It should be noted that the transformation of the logical view does not result in any data migration.

2.1.2 Moving Old Data

Under the logical view, we divide the data segment into d_{i-1}/N_i *regions*, each of which consists of N_i rows. For different regions, the rules of data migration are consistent, so we only focus on one region. As shown in Fig. 2, the shaded area is a *moving parallelogram*, where the data blocks will be migrated from. The base of a moving parallelogram is $N_i - N_{i-1}$, and the height is N_i. Fig. 2 depicts the moving trace of each migrated block, whose disk number is changed while the row number is always fixed. The source locations of moving data blocks are on the intersecting areas of the moving parallelogram and old disks, and the destination locations are on new disks and are outside the moving parallelogram.

After data migration, in a region, only the moving parallelogram is empty. The intersection of the moving parallelogram and each disk has $N_i - N_{i-1}$ blocks. Therefore, each disk will hold N_{i-1} data blocks after migration, so that Xscale ensures a uniform data distribution. In a region, the data blocks to be migrated are on the intersecting area of the moving parallelogram and old disks, and the number is $(N_i - N_{i-1}) \times N_{i-1}$. There are d_{i-1}/N_i regions in the whole data segment, so the total number of migrated data blocks is $(d_{i-1}/N_i) \times ((N_i - N_{i-1}) \times N_{i-1}) = (N_i - N_{i-1}) \times ((d_{i-1} \times N_{i-1})\ /N_i)$, and it is exactly the minimal number of data blocks to be moved. Therefore, Xscale minimizes data migration while maintaining a uniform data distribution across all the disks.

2.1.3 Stripe Reorganization

We suppose that each disk consists of s blocks. In an X-code array with N_i disks, the *coded array* is of size $N_i \times N_i$, where the first $N_i - 2$ rows contain regular data, and the last two rows contain parity blocks. Using the formula $p_i = s/N_i \times 2$, we can calculate the number of parity rows, which appears in an obviously negative correlation with the number of disks. With new disks added, the scale of the coded array increases, so the size of the parity rows set decreases. Xscale always chooses the last $(p_{i-1} - p_i)$ rows in the parity segment to turn into data rows.

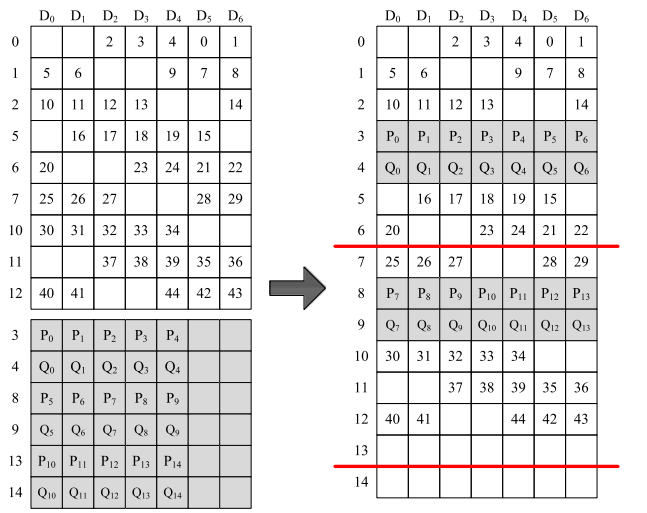


Fig. 3. Xscale's stripe reorganization during RAID-6 scaling from 5 to 7 disks. Here, $s = 15$, $d_{i-1} = 9$, $p_{i-1} = 6$, and $d_i = 10$, $p_i = 4$, so the last two rows in the parity segment turn into data rows.

To understand how Xscale performs stripe reorganization, we take RAID-6 scaling from 5 to 7 disks as an example. As shown in Fig. 3, newly added disks are inserted after Disk 4. Here, given $s = 15$, we have $d_{i-1} = 15/5 \times 3 = 9$, $p_{i-1} = 15/5 \times 2 = 6$, and $d_i = 15/7 \times\ 5 = 10$, $p_i = 15/7 \times 2 = 4$. So, the last $(p_{i-1} - p_i) = 6 - 4 = 2$ rows (i.e., Rows 13 and 14) in the parity segment turn into data rows. In this case, parity rows 3 and 4 protect data rows 0, 1, 2, 5, and 6. Parity rows 8 and 9 protect data rows 7, 10, 11, 12, and 13. Since there is no parity information protecting data row 14, this data row is not used for storing data.

During stripe reorganization, Xscale reconstructs the rule on how parity rows protect data rows in a coded array. We consider this issue in the logical view, with one data segment of d_i rows, and one parity segment of p_i rows. The data segment is divided into $d_i/(N_i - 2)$ *data pieces*, and the parity segment is divided into $p_i/2$ *parity pieces*. A data piece and a parity piece with the same ordinal number are paired into a coded array, where $(N_i - 2)$ data rows are protected by 2 parity rows.

2.1.4 Placing New Data

After RAID scaling, new data can be filled gradually, and we should determine an order of data placing, so that we can make rules showing how to map a logical address to its corresponding physical location.

As shown in Fig. 4, after the ith scaling operation, Xscale divides the RAID into three areas, according to B_i and B_{i-1}. Here, B_i is the position, below where the parity rows turn into data rows during the ith scaling. In these three areas, the rules of data placing are as follows.

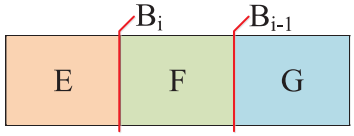


Fig. 4. Xscale divides the RAID into three areas for data placing according to B_i and B_{i-1}. B_i is the position, below where the parity rows turn into data rows during the ith scaling.

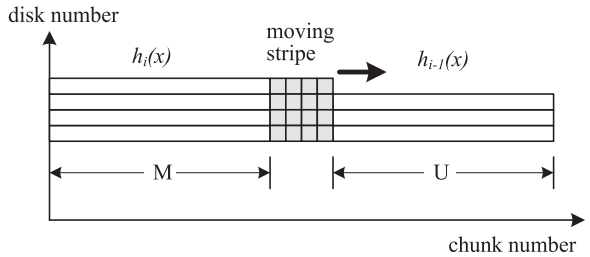


Fig. 5. Xscale's scaling workflow. "M": data blocks are migrated and parity blocks are updated; "U": data are not migrated. At any time, only the data in the moving stripe is reorganized.

- In area E, the number and the positions of data or parity rows in the RAID remain unchanged, during the ith scaling operation. Xscale skips the parity rows, and places new data blocks into the empty positions on the data rows, in a round-robin order. On each data row, there are $N_i - N_{i-1}$ empty positions, from where the data blocks are moved to the newly added disks.
- Similarly, in area F, there are $N_i - N_{i-1}$ empty positions on each data row. However, the parity rows will turn into data rows during the ith scaling operation. Each of these rows has N_i positions available for filling data, and Xscale places new data blocks from left to right. As for other rows, Xscale places data in a round-robin order.
- In area G, the parity rows have turned into data rows after previous scaling operations. It is a pure-data area. There are $N_i - N_{i-1}$ positions for filling data on each row. Xscale places new data blocks into these positions row by row, in a round-robin order.

2.2 Eliminating Metadata Updates

While data migration is in progress, the RAID storage serves user requests. Furthermore, the incoming user I/Os may write requests to migrated data. So it is necessary to write metadata to checkpoint the progress of data reorganization. Otherwise, data consistency may be destroyed, and if the system crashes during data reorganization, we cannot recover the data reorganization after the system restarts.

2.2.1 Existing Approaches

Synchronous metadata updates. Ordered operations of reorganizing a data stripe and updating the mapping metadata (a.k.a., *checkpoint*) can ensure data consistency. However, ordered operations require frequent metadata writes, and increases the cost of data reorganization significantly.

Lazy metadata updates. The technology of *lazy metadata updates* is used by MiPiL [22] in RAID-5 scaling. Data blocks are copied to new locations, and parity blocks are updated continuously. Mapping metadata is only updated when a user write request arrives in the area, where the data has been moved and the movement hasn't been checkpointed.

Lazy metadata updates can reduce the number of metadata writes significantly during RAID-5 scaling. Unfortunately, using lazy metadata updates in X-code RAID-6 scaling will result in data loss in the events of system crashes and disk failures. The reason is that one cannot identify whether a data block belongs to an old stripe or a new stripe.

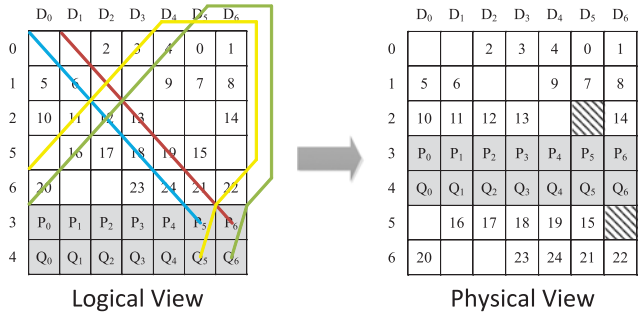


Fig. 6. Data reorganization within a moving stripe during RAID-6 scaling from 5 to 7 disks. Here, Block 2 on Disk 5, and Block 5 on Disk 6 are not used for storing user data.

2.2.2 How Xscale Eliminates Metadata Updates

Xscale uses a by-product of X-code RAID-6 scaling to identify migration boundaries, and therefore eliminates metadata updates without compromising data consistency and data reliability.

Basic operations for RAID scaling. Before performing data reorganization for RAID scaling, new disks are zeroized. This zeroizing operation does not take up the scaling time.

Fig. 5 illustrates an overview of the reorganization process. We use $h_i(x)$ to describe the geometry after the ith scaling operation, where N_i disks serve user requests. Within the moving stripe, data blocks are copied to new locations. When a user request arrives, if its physical block address is above the moving stripe, it is mapped with $h_{i-1}(x)$. If its physical block address is below the moving stripe, it is mapped with $h_i(x)$.

In the process of reorganizing the moving stripe, its parity blocks are also updated so as to guarantee data reliability. In a reorganized stripe, there usually exist non-zero bytes on the parity blocks in the new disks. Contrarily, in a stripe that has not been reorganized, all bytes in the new disks are zero because of the zeroizing operation. Checking the bytes on the parity blocks in the new discs makes it easy to tell whether a stripe has been reorganized. However, there is still a low probability that a reorganized stripe contains no non-zero bytes on the parity blocks in the new disks. To avoid the false negative, Xscale writes a two-byte magic number (i.e., *0XFFFF*) onto a specific location in each new disk to mark the reorganized stripe. The specific location is in an unused block during this scaling operation. Fig. 6 demonstrates data reorganization within a moving stripe during RAID-6 scaling from 5 to 7 disks. Here, Block 2 on Disk 5, and Block 5 on Disk 6 are not used for storing user data during the scaling operation. Therefore, they are available for the specific locations of the magic number.

After data reorganization in the moving stripe is completed, the moving stripe slides ahead by one stripe size. In this way, the newly added disks are gradually available to serve user requests. The whole process of RAID-6 scaling is completed when the moving stripe reaches the end of the RAID volume. From then on, the address mapping of the whole volume is performed through the new mapping function, $h_i(x)$, and N_i disks are used to serve requests.

Identifying reorganization boundary after accidental power-off. If the system detects an unfinished scaling operation in the booting stage, it continues the scaling process from the point of interruption. Due to the fact that Xscale performs data

Algorithm: Identifying($stripes, n$).

Input: The input parameters are $stripes$, n, where
$stripes$: the states of new stripes;
n: the total number of prospective new stripes.
Output: The ordinal number of the last reorganized stripe.

```
left ← 0; right ← n − 1; idx ← −1;                    (1)
while left ≤ right do                                  (2)
    mid ← (left + right)/2;                            (3)
    if Reorganized(stripes[mid]) then                  (4)
        idx ← mid;                                     (5)
        left ← mid + 1;                                (6)
    else                                               (7)
        right ← mid − 1;                               (8)
return idx.                                            (9)
```

Fig. 7. The Identifying algorithm used in Xscale.

reorganization in a strict one-by-one sequence, it is required to identify the reorganization boundary. In other words, Xscale needs to locate the most recently-reorganized stripe, which was being reorganized at the moment of the accidental power-off.

Xscale uses a modified *binary search* algorithm (see Fig. 7) to quickly find the most recently reorganized stripes in a stripe sequence. In the Identifying algorithm, the *Reorganized* function is used to determine whether an underlying stripe has been reorganized. First, Xscale checks the $2 \times (N_i - N_{i-1})$ parity blocks in the new disks. Only if one non-zero byte is found on those parity blocks, has this stripe been reorganized definitely, and the function returns *TRUE*. It is very simple to address those blocks according to the Xscale's data layout. For Stripe i, they are Block $i \times N_0 + (N_0 - 2)$ and Block $i \times N_0 + (N_0 - 1)$ in each new disk. Second, if no non-zero byte is eventually found, Xscale examines whether a two-byte magic number (i.e., 0XFFFF) is written on the specific location in each new disk. If so, this stripe has been reorganized. Otherwise, it is not reorganized yet.

2.2.3 Remarks

For a stripe that has been reorganized, only in a few cases does no non-zero byte exist on the $2 \times (N_i - N_{i-1})$ parity blocks in the new disks. Therefore, it is in a very low probability that Xscale writes a magic number (i.e., 0XFFFF) on the specific location in each new disk. Even if a magic number has to be written, the specific location is close to a data block that has been migrated. Xscale piggybacks the write of the magic number during data migration. As a result, the overhead of writing the magic number is negligible.

Even if the system fails unexpectedly during scaling, the reorganization boundary can be identified after rebooting, and therefore the data consistency is not destroyed. The Identifying algorithm of Xscale has time complexity $O(\log n)$ for n prospective new stripes.

3 Detailed Design

This section elaborates on Xscale's design details by discussing its mapping from a logical address to the corresponding physical address in a RAID-6 system.

Algorithm: Addressing(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;
s: the number of blocks in one disk;
x: a logical block number.
Output: The output parameters are d and b, where
d: the disk holding block x;
b: the physical block number on disk d.

```
t0 ← No. of scaling ops after which x is added;        (1)
(d, b) ← Placing(t0, N, s, x);                         (2)
b′ ← Phy2Log(t0, N, b);                                (3)
for i = t0 → t − 1 do                                  (4)
    d ← Moving(i + 1, N, d, b′);                       (5)
    b′ ← GoAhead(i + 1, N, b′);                        (6)
return (d, b).                                         (7)
```

Fig. 8. The addressing algorithm used in Xscale.

3.1 The Addressing Algorithm

Fig. 8 shows the *Addressing* algorithm to minimize data migration required by RAID-6 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 ith scaling operation, the RAID consists of $N[i]$ disks. Suppose that new disks are always inserted after the last disk in the RAID.

First, the *Addressing* algorithm determines that, block x is added after which scaling operation, denoted as t_0 (line 1), and uses the *Placing* function to get the location of block x in the physical view (line 2), when the block is just added into the RAID. It is important to note that the physical block number of block x will not change after scaling operations, because Xscale only moves data blocks within the same row. Second, the *Addressing* algorithm uses the *Phy2Log* function to calculate the corresponding logical row number of block x (line 3), according to the physical block number. Finally, with the functions *Moving* and *GoAhead*, the *Addressing* algorithm simulates the subsequent scaling operations, depicts the moving trace of block x in the logical view, and eventually locates the disk holding block x, after t scaling operations (lines 4-6). Thus, Xscale determines the location in the physical layout of RAID for a logical block number, x.

The *Placing* function. After the ith scaling operation from N_{i-1} to N_i disks, the number of parity rows decreases, i.e., $p_i = s/N_i \times 2$, and the number of data rows can be also determined by $d_i = s - p_i$. So, there are $((N_i - N_{i-1}) \times d_{i-1}) + ((p_{i-1} - p_i) \times N_i)$ empty positions available for new data blocks. The *Placing* function makes the rule how new data blocks should be placed into these positions.

The *Phy2Log* function. Before data migration, the *Addressing* algorithm uses the *Phy2Log* function to construct a logical view by separating data rows from parity rows. According to the physical block number of block x given by the *Placing* function, Xscale subtracts the number of parity rows, above the row where block x falls on, from the physical block number, and obtains the corresponding logical row number of block x.

Algorithm: Moving(i, N, d, b').

Input: The input parameters are i, N, d, and b', where
i: the number of scaling times;
N: the scaling history;
d: the disk number before a block is moved;
b': the row number in the logical view.
Output: The disk number after a block is moved.

```
m ← N[i − 1]; //the number of old disks                          (1)
n ← N[i] − m; //the number of new disks                          (2)
v_l ← (b' × n) mod (m + n);                                      (3)
v_r ← (v_l + n − 1) mod (m + n);                                 (4)
if ((d + m + n − v_l) mod (m + n) < n) then                      (5)
    if (m > n) then                                              (6)
        if (v_r ≥ m) then                                        (7)
            return d + n;                                        (8)
        else if (v_l > v_r) then                                 (9)
            return d + m;                                        (10)
        else                                                     (11)
            return d − v_l + m;                                  (12)
    else                                                         (13)
        if (v_r > v_l) then                                      (14)
            return d + n;                                        (15)
        else if (v_r < m) then                                   (16)
            return d + m;                                        (17)
        else                                                     (18)
            return d + v_r + 1;                                  (19)
```

Fig. 9. The moving function used in the addressing algorithm.

The *Moving* function. During the ith scaling operation, Xscale divides the data segment into d_{i-1}/N_i regions, and we only focus on the region that block x falls on. Then, Xscale identifies the location of block x in this region, with the logical row number and the disk number of block x. According to the migration rule shown in Section 2.1.2, Xscale determines how block x should be moved.

The *GoAhead* function. This function is used to obtain the logical row number of a physical row for the $(i+1)$th scaling operation from its logical row number for the ith scaling operation. After the ith scaling operation, suppose that the logical row number of block x is denoted as b'. According to the scaling history, Xscale can get the number of disks, and can further calculate the number of parity rows, after each time of scaling operation. Therefore, Xscale can calculate the number of the parity rows, which are above the corresponding physical row of b', turning into data rows in the $(i+1)$th scaling operation. Thus, Xscale obtains the row number in the logical view, after the $(i+1)$th scaling operation.

3.2 Moving Old Data

As shown in Fig. 9, the code of lines 1-5 is used to decide whether data block x will be moved during a RAID-6 scaling. In Fig. 2, there is a moving parallelogram in each region. The base of the parallelogram is n, and the height is $m+n$. If and only if, a data block is within the moving parallelogram, will it be moved. In the logical view, the row number of block x is b'. Row b' intersects the moving parallelogram at a segment, referred to as a *moving segment*. If the disk number d is within the moving segment, block x is within the moving parallelogram, and therefore it will be moved (line 5). Once a data block is determined to be moved, Xscale changes its disk number with the migration rules given in the *Moving* function (lines 6-19).

As shown in Fig. 11, n disks are newly added into a RAID made up of m old disks, so the moving segment has a width of n, and its two end points are v_l and v_r. If a block x has a disk number d, and it lies in a moving segment, it should be moved to a location on one newly added disk, denoted as d'. According to the values of m and n, we have a classified discussion. When $m > n$, the rules of data migration can be summarized into three cases as follows.

- The moving segment holding block x is disjoint with the new disks, i.e., $v_l < v_r < m$. The whole moving segment will be moved to the new disks, and the distance between block x and the left end of the moving segment will not change. In Fig. 11a, the equation $L_1 = L'_1 = d - v_l$ is satisfied, so we can locate the new disk holding block x after migration, with the formula $d' = L'_1 + m = d - v_l + m$ (line 12 in Fig. 9).
- The right end of the moving segment holding block x intersects with the new disks, i.e., $v_r \geq m$. As shown in Fig. 11a, the shaded area represents the intersection, and except for it, the moving segment will be moved. As in the previous case, $L_2 = L'_2 = d - v_l$ is satisfied, and the new disk number can be calculated with the formula $d' = L'_2 + v_r = d - v_l + v_r = d + n$ (lines 7-8 in Fig. 9).
- The left end of the moving segment holding block x intersects with the new disks, i.e., $v_l > v_r$. As shown in Fig. 11a, excluding the intersection, the moving segment will be moved. Similarly, this case satisfies the equation $L_3 = L'_3 = d$, and the new disk number can be calculated with the formula $d' = L'_3 + m = d + m$ (lines 9-10 in Fig. 9).

Similarly, when $m <= n$, the rules of data migration can also be shown in Fig. 11b, and can be described formally in Fig. 9 (lines 14-19).

3.3 Placing New Data

During the ith scaling operation, if block x is at a newly-added location, it is addressed via the *Placing* function given in Fig. 10. When a RAID is constructed from scratch (i.e., $i = 0$), it keeps a regular data layout of X-code, so block x can be simply placed into the RAID in a round-robin order (lines 1-3).

Let us examine the ith ($i > 0$) scaling operation, where n disks are added into the RAID (line 5). According to the scaling history, we can track the number of parity rows, and mark out the positions of B_i and B_{i-1} (line 4), below where the parity rows turn into data rows, after the ith and $(i-1)$th scaling, respectively. Among the newly added data blocks during the ith scaling, x is the yth block (line 9). With the value of y, we can determine the relative position between block x and the boundary lines drawn by B_i and B_{i-1}. Using the rules of data placing shown in Section 2.1.4, we can place block x into the RAID, according to the area on which it falls (lines 10-15).

Algorithm: Placing(i, N, s, x, d, b).

Input: The input parameters are i, N, s, and x, where
i: the number of scaling times;
N: the scaling history;
s: the number of blocks in one disk;
x: a logical block number.
Output: The output parameters are d and b, where
d: The disk holding block x;
b: the physical block number on disk d.

if $(i = 0)$ **then** (1)
  $(d, b) \leftarrow$ *data layout of regular X-code*; (2)
  return (d, b); (3)
calc B_i, B_{i-1}; (4)
$n \leftarrow (N[i] - N[i-1])$; //the number of new disks (5)
$old \leftarrow$ *No. of data blocks before the ith scaling*; (6)
$new \leftarrow$ *No. of data blocks added by the ith scaling*; (7)
$below \leftarrow$ *No. of data blocks added below* B_{i-1}; (8)
$y \leftarrow x - old$; (9)
if $(y < B_i \times n)$ **then** (10)
  $(d, b) \leftarrow$ *the placing rule in area E*; (11)
else if $(y \geq new - below)$ **then** (12)
  $(d, b) \leftarrow$ *the placing rule in area G*; (13)
else (14)
  $(d, b) \leftarrow$ *the placing rule in area F*; (15)
return (d, b). (16)

Fig. 10. The placing function used in the addressing algorithm.

3.4 Remarks about the Addressing Algorithm

The *Addressing* algorithm of Xscale is very simple and elegant, and requires very little code.

When a disk array boots, the RAID topology needs to be obtained from disks. The *Addressing* algorithm depends on the number of disks added during each scaling. If a RAID-6 array is scaled t times, Xscale needs to store $t + 1$ integers $(N[i], 0 \leq i \leq t)$ in a persistent storage.

4 Theoretical Proofs of Properties

For a RAID-6 scaling operation, it is desirable to ensure an even load distribution on all disks and minimal block movement. Furthermore, 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. Finally, if the system fails unexpectedly during scaling, the reorganization boundary can be quickly identified after rebooting. In this section, we formally prove that Xscale satisfies these requirements.

Theorem 4.1. *Xscale maintains a uniform data distribution after each RAID scaling.*

Proof Sketch. Assume that there are N_{i-1} old disks and $N_i - N_{i-1}$ new disks during a RAID scaling. As shown in Fig. 2, the physical data layout of disks is transformed into the logical view by Xscale's splitting operation, where data blocks are separated from parity blocks. Under the logical view, we divide the data segment into different regions, each of which consists of N_i rows. For different regions, the rules of data migration are completely identical. Therefore, it suffices to show that Xscale maintains a uniform data distribution in each region after this RAID scaling.

Before this RAID scaling, there are N_i data blocks on each of the N_{i-1} old data disks. As shown in Fig. 2, the intersection of the moving parallelogram and each disk has $N_i - N_{i-1}$ data blocks. Therefore, each old disk has $N_i - (N_i - N_{i-1}) = N_{i-1}$ data blocks after this scaling.

According to the Xscale moving scheme, no data blocks are moved onto the moving parallelogram. So, each new disk holds at most $N_i - (N_i - N_{i-1}) = N_{i-1}$ data blocks after this scaling. All new disks can hold at most $N_{i-1} \times (N_i - N_{i-1})$ data blocks. Since each old disk contributes $N_i - N_{i-1}$ blocks to the new disks, $N_{i-1} \times (N_i - N_{i-1})$ data blocks are put onto new disks. Consequently, any location on new disks outsides the moving parallelogram holds a data block. Therefore, each new disk has $N_i - (N_i - N_{i-1}) = N_{i-1}$ data blocks after this scaling. Each disk, either old or new, has N_{i-1} data blocks. That is to say, Xscale regains a uniform data distribution.

Theorem 4.2. *Xscale performs the minimum number of data migration during each RAID scaling.*

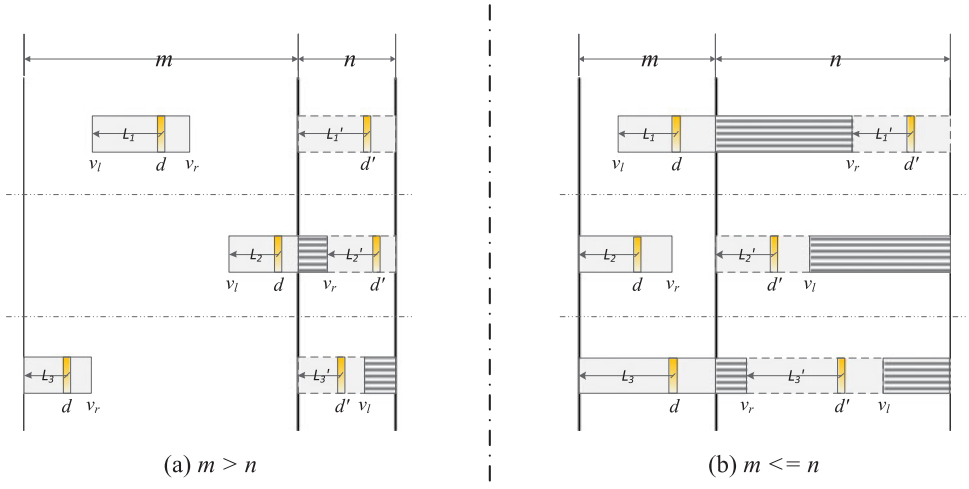


Fig. 11. Xscale's rules of data migration, during the scaling where n new disks are added into a RAID made up of m old disks. The solid-line rectangles represent the moving segments, where data blocks are moved from. The dotted rectangles represent the locations in the new disks, on which the moved blocks fall after migration.

TABLE 1
Comparison of Uniformity of Data Distribution

%	+2	+4	+2	+4	+2	+4
Round-Robin			0			
Semi-RR	29.7	42.8	14.6	24.3	9.2	16.3
Xscale			0			

TABLE 2
Comparison of Data Migration Ratio

%	+2	+4	+2	+4	+2	+4
Round-Robin			≈100.0			
Semi-RR	28.6	36.4	15.4	23.5	10.5	17.4
Xscale						

Proof Sketch. Assume that there are N_{i-1} old data disks and $N_i - N_{i-1}$ new data disks during a RAID scaling. Again, it suffices to show that Xscale performs the minimum number of data migrations in each region during this RAID scaling.

To maintain a uniform data distribution, the minimum number of blocks to be moved is $(N_{i-1} \times d_{i-1}) \times (N_i - N_{i-1}) / N_i$, where each old data disk has d_{i-1} data blocks. For one region, each old disk has N_i data blocks. Therefore, the minimum number of blocks to be moved for one region is $(N_{i-1} \times N_i) \times (N_i - N_{i-1})/N_i = N_{i-1} \times (N_i - N_{i-1})$.

As shown in Fig. 2, the intersection of the moving parallelogram and each disk has $N_i - N_{i-1}$ data blocks. In other words, each old disk contributes $N_i - N_{i-1}$ blocks to the new disks. In total, $N_{i-1} \times (N_i - N_{i-1})$ data blocks are moved onto new disks, which is exactly the minimum number of blocks to be moved.

Theorem 4.3. *The Addressing algorithm of Xscale has time complexity $O(t)$ after t RAID scalings.*

Proof Sketch. The Addressing algorithm is iterative, iterating no more than t times, namely, the number of scaling operations performed. Note that the Placing, Phy2Log, Moving, and GoAhead functions all take constant time, so the computation in each iteration is essentially constant time. Therefore, the Addressing algorithm has time complexity $O(t)$.

Theorem 4.4. *The Identifying algorithm of Xscale has time complexity $O(\log n)$ for n prospective new stripes.*

Proof Sketch. The identifying algorithm is a binary search for a specific circumstance. The time complexity of binary search is $O(\log n)$.

5 Property Examination

The purpose of this section is to quantitatively characterize the properties of Xscale—uniform data distribution, minimal data migration, fast data addressing, and eliminating metadata updates. We compare Xscale with the round-robin approach [16], [18], [19] and the Semi-RR approach [15] via simulation experiments. ALV [20], MDM [17] and FastScale [21] cannot be used in RAID-6, so they are not compared. An X-code [14] array is defined by a controlling parameter N, which must be a prime number greater than 2. From a 5-disk X-code array, we add two, four, two, four, two, four disks in turn using the three approaches respectively, and each disk holds $2 \times 1{,}024^2$ blocks. Our experiments simulate these scaling operations.

5.1 Uniform Data Distribution

In a region, we use the coefficient of variation (CV) of the numbers of blocks on different disks as a metric to quantify the uniformity of data distribution. The coefficient of variation is defined as the ratio of the standard deviation to the mean. The smaller the coefficient of variation is, the more uniform the data distribution of the region is. As for the whole RAID-6 system, we use the mean coefficient of variation (MCV) (i.e., the average of CVs in all the regions), to evaluate the uniformity across all the disks.

Table 1 shows the mean coefficient of variation versus the number of newly added disks in each scaling operation. For both the round-robin and Xscale approaches, the mean coefficients of variation remain at 0 percent as the times of disk additions increases. On the other hand, the Semi-RR approach causes excessive oscillation in the mean coefficient of variation, and the maximum can even reach 42.8 percent. This indicates that Xscale maintains a uniform data distribution after each RAID-6 scaling operation, while Semi-RR approach fails. So, Semi-RR causes non-uniform data distribution, also referred to as *data skew*. Data skew makes the runtime of application depend on the characteristic of access, so that the storage system is prone to exhibit an imbalance load and low I/O performance.

5.2 Minimal Data Migration

Table 2 shows the migration fraction (i.e., the fraction of data blocks to be migrated) versus the number of newly added disks in each scaling operation. Using the round-robin approach, the migration fraction is almost 100 percent, which causes a large migration cost.

The migration fractions are identical when using Semi-RR and Xscale approaches, and they are significantly smaller than the migration fraction in the round-robin approach. Assume that there are N_{i-1} old disks and $N_i - N_{i-1}$ new disks during a RAID-6 scaling operation. To regain a uniform data distribution, the minimal number of blocks to be moved is $(N_i - N_{i-1}) \times ((d_{i-1} \times N_{i-1})/N_i)$, where each old data disk has d_{i-1} data blocks. Our numerical analysis indicates that the migration fractions using Semi-RR and Xscale approaches reach this lower bound. In other words, Xscale minimizes data migrations during each RAID-6 scaling operation. Compared with the round-robin approach, Xscale reduces the number of data blocks to be migrated by 63.6-89.5 percent.

It should be noted that nearly all of the data has to be read by Xscale for recalculation of dual parity during each migration process.

5.3 Fast Data Addressing

In a scaled RAID, we run different approaches to locate all data blocks in a sequential order. We time the process, and calculate the average addressing time for a block, to quantify the calculation overheads. The testbed is an Intel Xeon CPU E5-2620 2.00 GHz machine with 32 GB of memory.

TABLE 3
Comparison of Addressing Time

μs	+2	+4	+2	+4	+2	+4
Round-Robin	0.014	0.014	0.014	0.013	0.013	0.013
Semi-RR	0.029	0.034	0.042	0.049	0.057	0.063
Xscale	0.044	0.052	0.067	0.077	0.092	0.101

TABLE 4
Comparison of the Number of Disk I/Os for Metadata Updates

$\times 10^5$	+2	+4	+2	+4	+2	+4
Round-Robin	1.80	1.06	1.32	0.90	0.97	0.74
Semi-RR	3.00	1.91	1.61	1.23	1.10	0.91
Xscale			0			

Additionally, an Ubuntu 12.04 x64 operating system is installed.

Table 3 shows the addressing time versus the number of newly added disks in each scaling operation. The round-robin approach has a low calculation overhead below 0.02 μs. Among the three approaches, Xscale has the largest overhead of about 0.1 μs, and the calculation overhead using Xscale shows an upward trend along with the times of disk additions, so there may be degradation of access performance after many scaling operations. On this issue, we conduct additional experiments to scale for more times, to see how Xscale behaves after the long-term scalability. There are initially 5 disks in the RAID. After 10 instances of scaling, there are 73 disks in the RAID, and the average addressing time is about 0.12 μs. After 15 instances of scaling, there are 199 disks in the RAID, and the average addressing time is about 0.14 μs. After 20 instances of scaling, there are 499 disks in the RAID, and the average addressing time is about 0.15 μs. In general, the addressing time slowly increases with the scaling times, and it is negligible compared to milliseconds of disk I/O time.

5.4 Eliminating Metadata Updates

Existing approaches to RAID-6 scaling require metadata updates to checkpoint the last reorganized stripe, to ensure data consistency and data reliability across system crash and/or disk failures. Metadata updates will cause disk I/O operations, and increase the cost of stripe reorganization.

Table 4 shows the number of disk I/Os for metadata updates versus the number of newly added disks in each scaling operation. Xscale piggybacks the write of the magic number with the regular data, so it requires no extra disk I/O to write metadata. Contrarily, round-robin and Semi-RR need to write metadata to the disks to checkpoint the migration progress. So, Xscale can eliminate metadata updates during RAID-6 scaling to improve the performance, while guaranteeing data consistency and data reliability.

6 Experimental Evaluation

In this section, we compare Xscale with round-robin, Semi-RR, and the SDM scheme [23] through detailed experiments. SDM is a stripe-based data migration scheme used to improve the scalability of RAID-6, and it also provides uniform data distribution and minimal data migration. In these experiments, each disk contains $512 \times 1{,}024$ blocks, and the size of a block is 32 KB, so each disk has a capacity of 16 GB.

6.1 Simulation System

To simulate the online RAID scaling, the simulator consists of two components: a workload generator and a disk array. According to trace files, the workload generator initiates online user requests at the appropriate time, so that particular workloads are induced on the disk array.

The disk array is made up of an array controller and a storage component. 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 storage component simulates modern disk drivers in great detail.

The simulator is implemented in SimPy [24] and DiskSim [25]. The workload generator and the array controller are implemented in SimPy. The storage component is 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 [26].

6.2 Application 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 [27]. It was collected with one client running 20 iterations.
- *Fin* is obtained from the Storage Performance Council (SPC) [28], a vendor-neutral standards body. The Fin trace was collected from OLTP applications [29] 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.

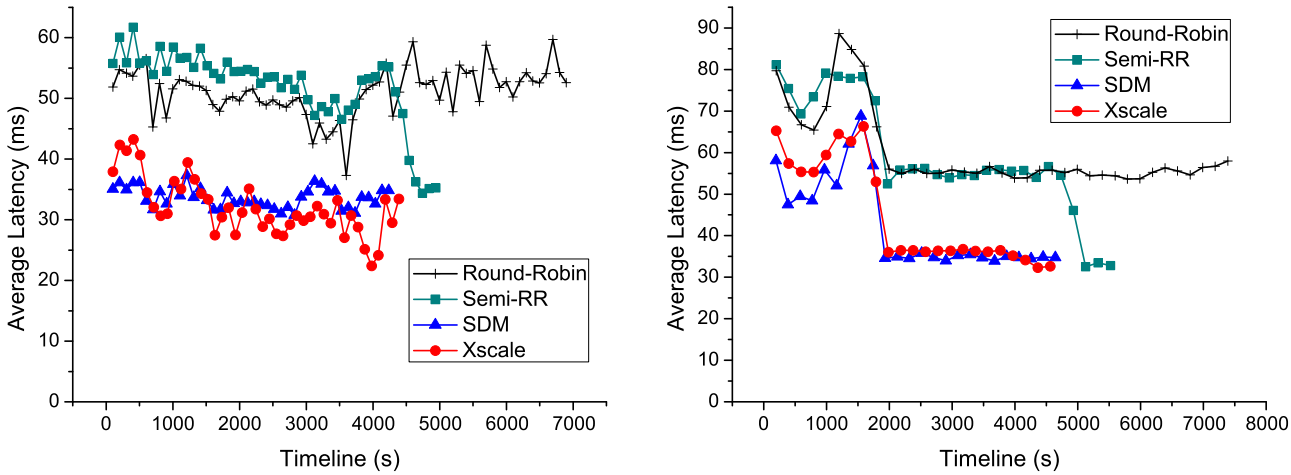
6.3 Experimental Results

6.3.1 The Scaling Efficiency

In this section, we focus on comparing reorganization times and user request latencies when different scaling programs are running in the background. We conduct a scaling operation of adding 2 disks to a 11-disk RAID, and each experiment lasts the duration of the data reorganization. We collect the I/O latencies of all user requests. We divide the I/O latency sequence into multiple sections according to I/O issuing time. From each section, we get a local average latency, i.e., the average of I/O latencies in a section.

Fig. 12a plots the local average latencies using the four approaches as the time increases along the x-axis, under the Fin workload. It illustrates that Xscale makes an improvement over round-robin and Semi-RR in two metrics.

First, the reorganization time using Xscale is shorter than that using Semi-RR and round-robin. They are 4,389 seconds, 5,045 seconds, and 6,897 seconds, respectively. Xscale has a 36.36 percent shorter reorganization time than round-



(a) under the Fin workload.

(b) under the TPC-C workload.

Fig. 12. Comparison of reorganization times and I/O latencies.

robin. The key factor in Xscale's reduced reorganization time is the significant decline of the amount of the data to be moved. When round-robin is used, almost all of data blocks have to be migrated. However, when Xscale is used, only 15.36 percent of data blocks require being migrated. Another technique, i.e., eliminating metadata updates, also helps reduce the number of disk I/Os during RAID scaling.

Second, the local average latencies of round-robin and Semi-RR are significantly longer than those of Xscale. The global average latencies using round-robin and Semi-RR respectively reach 50.60 and 52.18 ms, while that using Xscale is 31.50 ms. This is because round-robin requires more disk I/Os during stripe reorganization than Xscale, so that user request latencies are enlarged. For Semi-RR, its migration rule makes the reorganization process involve extra stripes beside the moving stripe, and this takes a great impact on application performance.

SDM's data reorganization is based on the "stripe set" (i.e., a set of stripes) as a unit, which is larger than Xscale's. This improves SDM's efficiency of data migration to some extent, but also brings a negative effect towards the response speed of disk array to user requests. The reorganization time using SDM is 4,237 seconds, which is slightly shorter than that of Xscale; but on the other hand, the global average I/O latency using SDM reaches 33.56 ms, which is longer than that of Xscale.

A factor that might affect the benefits of Xscale is the workload under which data reorganization performs. Under the TPC-C workload, we also perform the "11+2" scaling operation. Fig. 12b plots the local average latencies versus the reorganization times for the four approaches. It shows, once again, the efficiency of Xscale in improving the reorganization time. The reorganization times using round-robin, Semi-RR, and Xscale are 7,588, 5,521, and 4,761 seconds, respectively. Xscale brings an improvement of 37.26 percent in the reorganization time. Likewise, the local average latencies of Xscale are also obviously shorter than those of round-robin and Semi-RR. The global average latency using Xscale is 49.91 ms, while that using round-robin and Semi-RR reaches 65.06 and 63.92 ms. In addition, the reorganization time using SDM is 4,643 seconds, and it is slightly shorter than that of Xscale; however, the global average I/O latency using SDM reaches 50.94 ms, and it is longer than that using Xscale.

To make our observations more convincing and intuitive, we carry out experiments under different workloads, and summarize the results in Figs. 13 and 14. Fig. 13 shows a comparison in the reorganization time among round-robin, Semi-RR, SDM, and Xscale. Furthermore, we conduct a comparison experiment on the reorganization time with no application workload. To scale a RAID volume offline, round-robin uses 6653 seconds, whereas Xscale consumes only 4,237 seconds, so Xscale provides an improvement of 36.31 percent in the reorganization time. Fig. 14 shows a comparison in the global average I/O latency among the four scaling approaches. Under different workloads, Xscale saves the response time of user I/Os by 23.29-37.74 percent compared with round-robin, and saves the response time by 21.92-39.63 percent compared with Semi-RR.

From these results, we draw a conclusion. Under various application workloads, Xscale consistently outperforms round-robin by 35.62-37.26 percent in the reorganization time, with shorter response time of user I/Os. Xscale also outperforms Semi-RR by 21.92-39.63 percent in the response time of user I/Os, and requires shorter reorganization time.

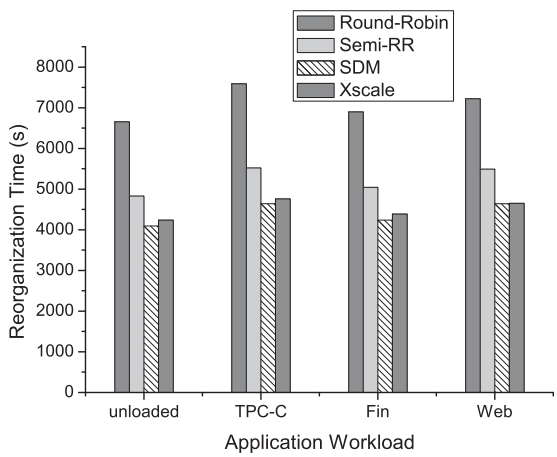


Fig. 13. Comparison of reorganization times under different workloads. The label "unloaded" means scaling a RAID volume offline.

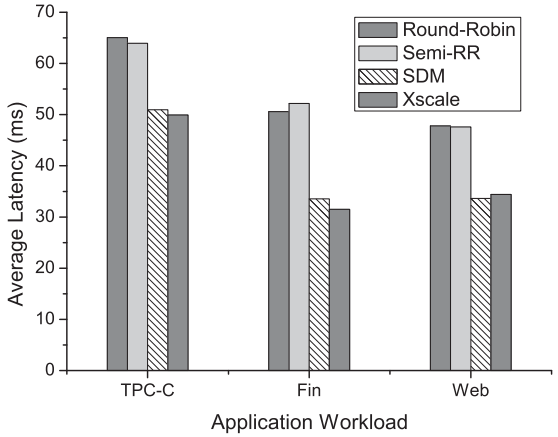


Fig. 14. Comparison of average I/O latencies under different workloads, while different scaling programs are running in the background.

Xscale and SDM have about equal scaling efficiency, in terms of both reorganization time and I/O latency. However, in the current state, SDM can be used in X-code RAID-6 scaling only one time, rather than multiple times.

6.3.2 The Performance after Scaling

The above experiments show that Xscale achieves good efficiency of RAID scaling. One of our concerns is whether there is a penalty in the performance of the data layout after scaling using Xscale, compared with the data layouts of round-robin and SDM.

We use the Fin workload to measure the performances of the three RAIDs, scaled from the same RAID using round-robin, SDM and Xscale. Each experiment lasts 3,600 seconds, and records the latency of each I/O. According to the issue time, the I/O latency sequence is divided into multiple sections evenly, and each section lasts about 100 seconds. Then we get a local average latency from each section.

First, we compare the performances of three RAIDs, after one scaling operation "7+4", using the three scaling approaches. Fig. 15a plots the local average latencies of the four RAIDs as the time increases along the *x*-axis. We find that the performances of the Xscale RAID and the SDM RAID are very close, and they both outperform the round-robin RAID. The global average latencies of round-robin, SDM, and Xscale are 5.35, 4.64, and 4.50 ms, respectively.

Second, we compare the performances of two RAIDs, after two scaling operations "7+4+2", using round-robin and Xscale. SDM has just considered one time of X-code scaling, so it cannot be a right comparison in this case. Fig. 15b plots the local average latencies of the two RAIDs as the time increases along the *x*-axis. It shows that the Xscale RAID slightly outperforms the round-robin RAID. The global average latencies of round-robin and Xscale are 4.65 and 4.32 ms, respectively.

These experimental results show that Xscale will not result in degradation of access performance, compared with round-robin and SDM. To substantiate this observation, we also measure the performances of the three RAIDs under the TPC-C workload.

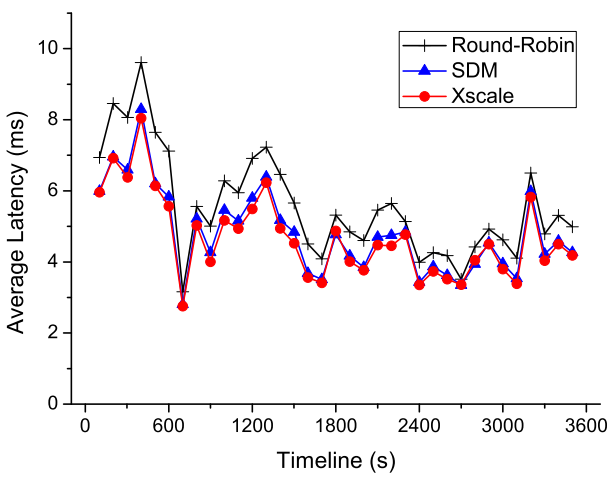
First, we compare the performances of three RAIDs, after one scaling operation "7+4", using the three scaling approaches. Fig. 16a plots the local average latencies of the three RAIDs as the time increases along the *x*-axis. We find that the performances of the three RAIDs are all very close. The global average latencies of round-robin, SDM, and Xscale are 27.54, 29.62, and 29.68 ms, respectively.

Second, we compare the performances of two RAIDs, after two scaling operations "7+4+2", using round-robin and Xscale. Fig. 16b plots the local average latencies of the two RAIDs as the time increases along the *x*-axis. It again reveals the approximate equality in the performances of the two RAIDs. The global average latencies of round-robin and Xscale are 29.33 and 28.03 ms, respectively.

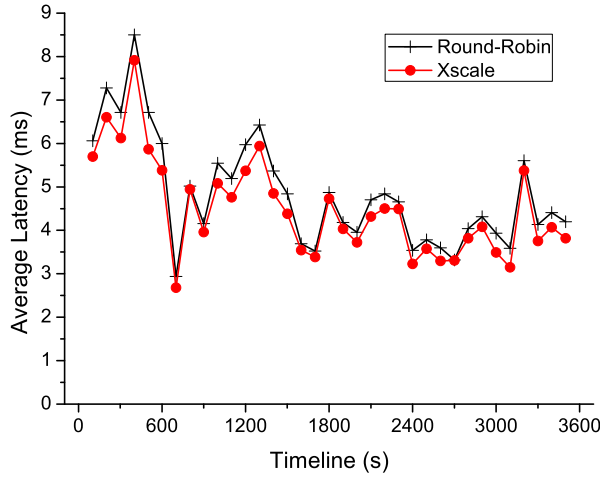
From the above experiments, we can draw a conclusion. There is no penalty in the performance of the data layout after scaling using Xscale, compared with the layouts maintained by round-robin and SDM. In fact, the access performance of the three RAIDs scaled using different approaches is almost equal.

7 Related Work

Efforts concentrating on RAID scaling approaches are divided into two categories, optimizing the process of data migration and reducing the number of data blocks to be moved.

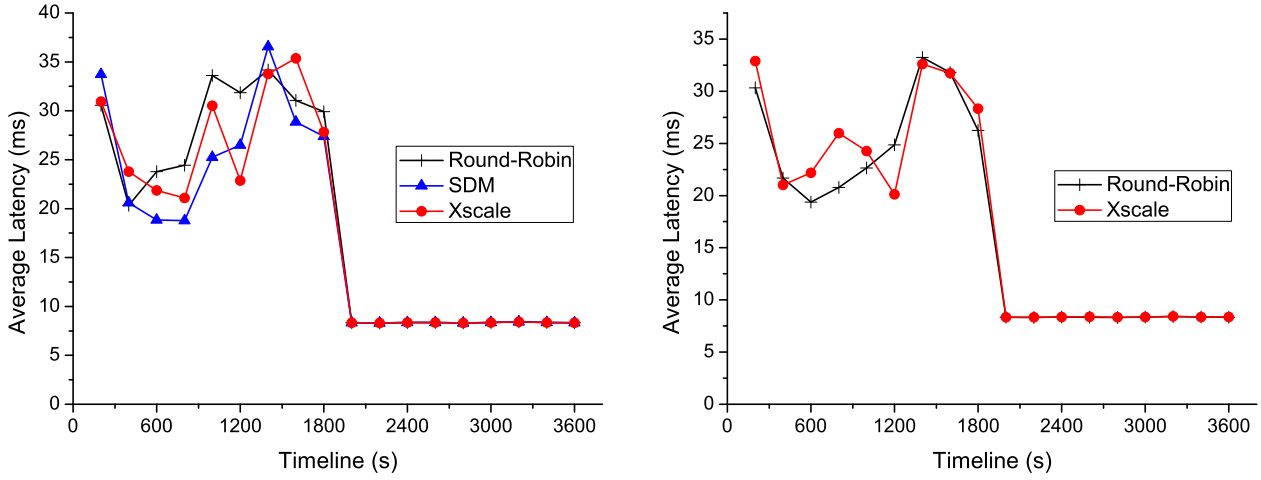


(a) After one scaling operation "7+4".



(b) After two scaling operations "7+4+2".

Fig. 15. Comparison of I/O latencies under the Fin workload.



(a) After one scaling operation "7+4". (b) After two scaling operations "7+4+2".

Fig. 16. Comparison of I/O latencies under the TPC-C workload.

7.1 Optimizing Data Migration for RAID Scaling

Conventional approaches redistribute data and preserve the round-robin order after adding disks. All data blocks are migrated in the scaling process. Brown [18] designed a reshape toolkit in the Linux kernel (MD-Reshape), which writes mapping metadata with a fixed-size data window. User requests to the 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.

Gonzalez and Cortes [16] proposed a gradual assimilation (GA) approach to control the overhead of expanding a RAID-5 system, but it has a large redistribution cost since all parities still need to be modified after data migration.

US patent #6000010 [30] presents a method to scale RAID-5 volumes, eliminating the need to rewrite data and parity blocks to the original disks. This, however, may lead to an uneven distribution of parity blocks and penalize write requests.

The MDM method [17] 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 is unable to increase (only keep) the storage efficiency by adding new disks.

Franklin and Wong [31] propose using 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. Similar to WorkOut [32], this kind of method requires spare disks to be available.

Zhang et al. [19], [20] 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 improve the efficiency of RAID-5 scaling. However, ALV still suffers from large data migration.

7.2 Reducing Data Migration for RAID Scaling

With the development of object-based storage, randomized RAID [15], [33], [34] reduces data migration while delivering a uniform load distribution. The cut-and-paste placement strategy [34] uses a randomized allocation strategy to place data across disks. Seo and Zimmermann [35] proposed an approach to find a sequence of disk additions and removals for the disk replacement problem. The goal is to minimize the data migration cost. The SCADDAR algorithm [15] uses a pseudo-random function to minimize the amount of data to be moved. RUSH [36], [37] and CRUSH [38] are two algorithms for online placement and reorganization of replicated data. The Random Slicing strategy [39] used a small table with information on insertion and removal operations to reduce the required randomness and deliver a uniform load distribution with minimal migration. These randomized strategies are designed for object-based storage systems, and focus only on how blocks are mapped to disks, ignoring the inner data layout of each individual disk.

There are several deterministic approaches to improve the extensibility of RAID. HP's AutoRAID [40] allows an online capacity expansion without data migration, by which newly created RAID volumes use all disks, and previously created ones use only the original disks.

To reduce data migration, the Semi-RR approach [15] modifies the round-robin scheme, and requires a block movement only if the target disk number is one of new disks. Semi-RR reduces data migration significantly. Unfortunately, it does not guarantee uniform distribution of data blocks after subsequent scaling operations. This will deteriorate the initial equally-distributed load.

The GSR approach [41] divides data on the original array into two sections, and moves the second one onto the new disks, keeping the layout of most stripes. Its main limitation is performance: after upgrades, accesses to the first section are served by original disks, and accesses to the second are served only by newer disks.

The SDM scheme [23] optimizes data movements according to the future parity layout, which minimizes the overhead of data migration and parity modification. It also provides uniform data distribution and minimal data migration. In the current state, however, SDM cannot address data newly added after one RAID scaling operation. Therefore, SDM can be used in X-code RAID-6 scaling only one time, rather than multiple times.

Zheng and Zhang [21] proposed the FastScale approach to RAID-0 scaling. FastScale minimizes data migration

while maintaining a uniform data distribution. FastScale provides a good starting point for RAID-6 scaling. However, RAID-6 scaling is more challenging, as discussed in Section 1.2.

RS6 [42] is a new approach to accelerating RDP [3] RAID-6 scaling by reducing disk I/Os and XOR operations. However, RDP is a horizontal code, and RS6 does not handle RAID-6 scaling with vertical codes.

Miranda and Cortes presented CRAID [43], which reorganizes only frequently-accessed data when new disks are added, and therefore, reduces this migration even further. Compared with Xscale, CRAID needs less data migration for RAID scaling, while it has to identify those frequently-accessed data all the time. Therefore, CRAID performs extra statistics of data accesses, and in turn suffers from additional spatial and temporal overheads. It should be noted that RAID scaling involves occasional events, while statistics of data accesses are performed all the time by CRAID.

8 Conclusions

In this paper, we propose Xscale – a new approach to accelerating X-code RAID-6 scaling by using lightweight data reorganization. While preserving the uniformity of data distribution, Xscale reaches the lower bound of the migration fraction. Furthermore, Xscale uses a by-product of X-code scaling to identify migration boundaries, and therefore eliminates metadata updates without compromising data consistency and data reliability.

To evaluate the benefits of our Xscale approach, we conduct extensive simulation experiments to quantitatively characterize the properties of Xscale, and compare it with round-robin, Semi-RR, and SDM. The experimental results show that Xscale achieves both good scaling efficiencies and low I/O latencies after scaling. Compared with the round-robin approach, Xscale reduces the number of blocks to be moved by 63.6-89.5 percent, saves the reorganization time by 35.62-37.26 percent, and reduces the I/O latency by 23.29-37.74 percent while different scaling programs are running in the background. Xscale also outperforms Semi-RR by 21.92-39.63 percent in the response time of user I/Os, and requires shorter reorganization time. SDM has about equal reorganization time and I/O latency with Xscale, but it only supports one time of X-code RAID-6 scaling in the current state. In addition, there is no penalty in the performance of the data layout after scaling using Xscale, compared with the layouts maintained by round-robin and SDM.

Acknowledgments

This work was supported by the National Grand Fundamental Research 973 Program of China under Grant No. 2014CB340402, the National Natural Science Foundation of China under Grants 61170008 and 61272055, and the National High Technology Research and Development Program of China under Grant No. 2013AA01A210.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (raid)," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1988, pp. 109–116.

[2] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Comput. Surv.*, vol. 26, pp. 145–185, Jun. 1994.

[3] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, 2004, pp. 1–14.

[4] E. Pinheiro, W.-D. Weber, and L. A. Barroso, "Failure trends in a large disk drive population," in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, pp. 17–29.

[5] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1,000,000 hours mean to you?," in *Proc. 5th USENIX Conf. File Storage Technol.*, 2007, pp. 1–16.

[6] K. Hwang, H. Jin, and R. Ho, "Raid-x: A new distributed disk array for i/o-centric cluster computing," in *Proc. 9th IEEE Int. Symp. High Perform. Distrib. Comput.*, 2000, pp. 279–286.

[7] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence, "Fab: Building distributed enterprise disk arrays from commodity components," in *Proc. 11th Int. Conf. Archit. Support Program. Lang. Operating Syst.*, 2004, pp. 48–58.

[8] X. Yu, B. Gum, Y. Chen, R. Y. Wang, K. Li, A. Krishnamurthy, and T. E. Anderson, "Trading capacity for performance in a disk array," in *Proc. 4th Conf. Symp. Operating Syst. Des. Implementation*, 2000, pp. 17–32.

[9] S. Ghandeharizadeh and D. Kim, "On-line reorganization of data in scalable continuous media servers," in *Proc. 7th Int. Conf. Database Expert Syst. Appl.*, 1996, pp. 751–768.

[10] D. A. Patterson, "A simple way to estimate the cost of downtime," in *Proc. 16th USENIX Conf. Syst. Admin.*, 2002, pp. 185–188.

[11] J. S. Plank, "The raid-6 liberation codes," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, pp. 1–14.

[12] C. Wu, S. Wan, X. He, Q. Cao, and C. Xie, "H-code: A hybrid mds array code to optimize partial stripe writes in raid-6," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2011, pp. 782–793.

[13] C. Wu, X. He, G. Wu, S. Wan, X. Liu, Q. Cao, and C. Xie, "Hdp code: A horizontal-diagonal parity code to optimize i/o load balancing in raid-6," in *Proc. IEEE/IFIP 41st Int. Conf. Dependable Syst. Netw.*, 2011, pp. 209–220.

[14] L. Xu and J. Bruck, "X-code: MDS array codes with optimal encoding," *IEEE Trans. Inf. Theory*, vol. 45, no. 1, pp. 272–276, Jan. 1999.

[15] A. Goel, C. Shahabi, S.-Y. D. Yao, and R. Zimmermann, "Scaddar: An efficient randomized technique to reorganize continuous media blocks," in *Proc. 18th Int. Conf. Data Eng.*, 2002, pp. 473–482.

[16] J. L. Gonzalez and T. Cortes, "Increasing the capacity of raid5 by online gradual assimilation," in *Proc. Int. Workshop Storage Netw. Archit. Parallel I/Os*, 2004, pp. 17–24.

[17] S. Hetzler, "Data storage array scaling method and system with minimal data movement," U.S. Patent App. 12/134,051, Nov. 6, 2008.

[18] N. Brown, "Online raid-5 resizing. drivers/md/raid5.c in the source code of linux kernel 2.6.18," Sep. 2006.

[19] G. Zhang, J. Shu, W. Xue, and W. Zheng, "Slas: An efficient approach to scaling round-robin striped volumes," *Trans. Storage*, vol. 3, no. 1, pp. 1–39, Mar. 2007.

[20] G. Zhang, W. Zheng, and J. Shu, "Alv: A new data redistribution approach to raid-5 scaling," *IEEE Trans. Comput.*, vol. 59, no. 3, pp. 345–357, Mar. 2010.

[21] W. Zheng and G. Zhang, "Fastscale: Accelerate raid scaling by minimizing data migration," in *Proc. 9th USENIX Conf. File Stroage Technol.*, 2011, pp. 149–161.

[22] G. Zhang, W. Zheng, and K. Li, "Rethinking raid-5 data layout for better scalability," *IEEE Trans. Comput.*, vol. 63, no. 11, pp. 2816–2828, Nov. 2014.

[23] C. Wu, X. He, J. Han, H. Tan, and C. Xie, "Sdm: A stripe-based data migration scheme to improve the scalability of raid-6," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2012, pp. 284–292.

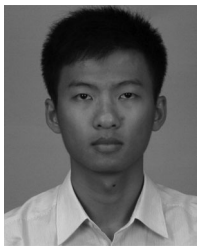
[24] K. G. Müller and T. Vignaux. (2009, Apr.). Simpy 2.0.1's documentation [Online]. Available: https://pypi.python.org/pypi/simpy/2.0.1

[25] J. S. Bucy, J. Schindler, S. W. Schlosser, G. R. Ganger, and Contributors, "The DiskSim Simulation Environment Version 4.0 Reference Manual," Parallel Data Lab., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-PDL-08-101, May 2008.

[26] (2001, Apr.). IBM Storage Products. Ultrastar 36Z15 Hard disk drive specifications [Online]. Available: http://www.hgst.com/tech/techlib.nsf/techdocs/85256AB8006A31E587256A7800739-FEB/$file/U36Z15_sp10.PDF. Revision 1.0

[27] (2010, Dec.). TPC-C, Postgres, 20 iterations, DTB v1.1 [Online]. Available: http://archive.is/tds.cs.byu.edu. Performance Evaluation Laboratory, Brigham Young University. Trace Distribution Center.
[28] (2010, Dec.). [Online]. Available: http://www.storageperformance.org/home
[29] (2007, Jun.). OLTP Application I/O and Search Engine I/O [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage. UMass Trace Repository.
[30] C. 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," U.S. Patent 6,000,010, Dec. 7, 1999.
[31] C. Franklin and J. Wong, "Expansion of raid subsystems using spare space with immediate access to new space," U.S. Patent 7,111,117, Sep. 19, 2006.
[32] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "Workout: I/O workload outsourcing for boosting raid reconstruction performance," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 239–252.
[33] J. R. Santos, R. R. Muntz, and B. Ribeiro-Neto, "Comparing random data allocation and data striping in multimedia servers," *SIGMETRICS Perform. Eval. Rev.*, vol. 28, pp. 44–55, Jun. 2000.
[34] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, distributed data placement strategies for storage area networks (extended abstract)," in *Proc. 12th Annu. ACM Symp. Parallel Algorithms Archit.*, 2000, pp. 119–128.
[35] B. Seo and R. Zimmermann, "Efficient disk replacement and data migration algorithms for large disk subsystems," *Trans. Storage*, vol. 1, pp. 316–345, Aug. 2005.
[36] R. J. Honicky and E. L. Miller, "A fast algorithm for online placement and reorganization of replicated data," in *Proc. 17th Int. Symp. Parallel Distrib. Process.*, 2003, pp. 1–10.
[37] R. J. Honicky and E. L. Miller, "Replication under scalable hashing: A family of algorithms for scalable decentralized data distribution," in *Proc. 18th Int. Parallel Distrib. Process. Symp.*, Apr. 2004, pp. 96–105.
[38] S. A. Weil, S. A. Brandt, E. L. Miller, and C. Maltzahn, "Crush: Controlled, scalable, decentralized placement of replicated data," in *Proc. ACM/IEEE Conf. Supercomput.*, 2006, pp. 122–133.
[39] 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," in *Proc. 18th Int. Conf. High Perform. Comput.*, 2011, pp. 1–10.
[40] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The hp autoraid hierarchical storage system," *ACM Trans. Comput. Syst.*, vol. 14, pp. 108–136, Feb. 1996.
[41] C. Wu and X. He, "Gsr: A global stripe-based redistribution approach to accelerate raid-5 scaling," in *Proc. 41st Int. Conf. Parallel Process.*, 2012, pp. 460–469.
[42] G. Zhang, K. Li, J. Wang, and W. Zheng, "Accelerate rdp raid-6 scaling by reducing disk i/os and xor operations," *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 32–44, Jan. 2015.
[43] A. Miranda and T. Cortes, "Craid: Online raid upgrades using dynamic hot data reorganization," in *Proc. 12th USENIX Conf. File Storage Technol.*, 2014, pp. 133–146.

Guangyan Zhang received the bachelor's and master's degrees in computer science from Jilin University in 2000 and 2003, respectively. He then went on to receive the doctorate degree in computer science and technology from Tsinghua University in 2008. He is now 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.



Guiyong Wu received the bachelor's degree in computer science and technology from Tsinghua University in 2014 and is currently woking toward the master's degree in the Department of Computer Science and Technology at Tsinghua University. His current research interests include network storage and distributed systems.

Yu Lu is now an undergraduate student in the Department of Computer Science and Technology at Tsinghua University. His current research interest is in network storage.

Jie Wu is the chair and a Laura H. Carnell professor in the Department of Computer and Information Sciences at Temple University. Prior to joining Temple University, he was a program director at the National Science Foundation and a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications.

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 covers distributed computing, compiler techniques, and network storage.