Rethinking RAID-5 Data Layout for Better Scalability

Guangyan Zhang, Weimin Zheng, and Keqin Li

Abstract—In RAID-5, data and parity blocks are distributed across all disks in a round-robin fashion. Previous approaches to RAID-5 scaling preserve such round-robin distribution, therefore requiring all the data to be migrated. In this paper, we rethink RAID-5 data layout and propose a new approach to RAID-5 scaling called MiPiL. First, MiPiL minimizes data migration while maintaining a uniform data distribution, not only for regular data but also for parity data. It moves the minimum number of data blocks from old disks to new disks for regaining a uniform data distribution. Second, MiPiL optimizes online data migration with piggyback parity updates and lazy metadata updates. Piggyback parity updates during data migration reduce the numbers of additional XOR computations and disk I/Os. Lazy metadata updates minimize the number of metadata writes without compromising data reliability. We implement MiPiL in Linux Kernel 2.6.32.9, and evaluate its performance by replaying three real-system traces. The results demonstrate that MiPiL consistently outperforms the existing "moving-everything" approach by 74.07-77.57% in redistribution time and by 25.78-70.50% in user response time. The experiments also illustrate that under the WebSearch2 and Financial1 workloads, the performance of the RAID-5 scaled using MiPiL is almost identical to that of the round-robin RAID-5.

Index Terms—Data migration, disk array, metadata update, parity update, RAID-5 scaling

1 Introduction

Via disk striping and rotated parity, RAID-5 [1], [2] achieves high performance, large capacity, and data reliability. As a result, it is widely used on servers. In recent years, user data grow explosively and computing powers enhance rapidly. Under this background, to increase the capacity and I/O performance of a RAID-5 storage system, scaling it up is a common method. First, by adding more disks, a RAID-5 volume provides larger storage capacity and higher I/O performance [3], [4]. This can meet the increasing requirements on storage systems in various online applications [4], while avoiding the extremely high downtime cost [5]. Second, RAID-based architectures are also used for clusters and large-scale storage systems, where scalability plays a significant role [6]-[8]. Such disk addition is termed "*RAID-5 scaling*".

Performing the process of RAID-5 scaling is a difficult technical challenge for two reasons. First, to regain uniform data distribution in all disks including old and new, a RAID-5 scaling requires certain blocks to be moved onto added disks. In other words, RAID-5 scaling means restriping RAID-5 when new disks are added. Second, in today's server environments, many applications access data constantly. The cost of downtime is extremely high [5], giving rise to the necessity of online and real-time scaling. Therefore, RAID-5 scaling requires an efficient approach to redistributing the data online with the following requirements. (1) Data redistribution should be completed in a short period of time. (2) The impact of data redistribution on application performance should not be significant. (3) Data reliability should be guaranteed even if the system crashes and/or one disk fails during the scaling process.

Existing approaches to RAID-5 scaling [9]-[11] are restricted by preserving a round-robin data distribution after adding disks. Therefore, 100 percent of data blocks need to be migrated. Large data migration results in expensive cost of a RAID-5 scaling. There are some efforts [9], [11] concentrating on optimization of data migration. They improve the performance of RAID-5 scaling to certain extent, but do not completely overcome the limitation of large data migration.

In this paper, we rethink RAID-5 data layout for better scalability and propose a new approach to RAID-5 scaling called MiPiL (*Mi*nimizing data migration, *Pi*ggyback parity updates, and *L*azy metadata updates). It accelerates RAID-5 scaling by minimizing data migration. MiPiL moves data blocks from old disks to new disks for regaining a uniform data distribution, while not migrating data among old disks. The migration fraction of MiPiL reaches the lower bound of the migration fraction for RAID-5 scaling. We design an elastic addressing function through which the location of one block can be easily computed without any lookup operation.

MiPiL has several unique features as follows.

- MiPiL maintains a uniform data distribution after RAID-5 scaling, not only for regular data but also for parity data.
- MiPiL minimizes the amount of data to be migrated during a scaling.
- MiPiL preserves simple management of data due to deterministic placement.

- *G. Zhang 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.*
- *K. Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561. E-mail: lik@newpaltz.edu.*

Manuscript received 22 Dec. 2012; revised 10 June, 2013; accepted 22 June, 2013. Date of publication 01 July 2013; date of current version 14 Oct. 2014. Recommended for acceptance by K. Li.
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/TC.2013.143

- MiPiL sustains the above three features after multiple disk additions.

MiPiL also optimizes online data migration with piggyback parity updates and lazy metadata updates. Piggyback parity updates during data migration reduce the numbers of additional XOR computations and disk I/Os. Lazy metadata updates minimize the number of metadata writes without compromising data reliability. Here, metadata is used to map a logical address as seen by the host to a physical location inside an array.

We implement MiPiL 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 MiPiL outperforms the existing "move-everything" approach by 74.07-77.57% in redistribution time and by 25.78-70.50% in user response time simultaneously. Our experiments also illustrate that under the WebSearch2 and Financial1 workloads, the performance of the RAID-5 scaled using MiPiL is almost identical to that of the round-robin RAID-5.

The rest of the paper is organized as follows. In Section 2, we present the design of MiPiL, including the design objectives, an illustration of how MiPiL works, and the addressing algorithm. In Section 3, we describe the optimization techniques of MiPiL, i.e., piggyback parity updates and lazy metadata updates. In Section 4, we give an implementation of MiPiL in a Linux Kernel. In Section 5, we demonstrate experimental results to show the performance of MiPiL. In Section 6, we review related research in the literature. We conclude the paper in Section 7.

2 The MiPiL Approach

We propose the MiPiL approach by rethinking RAID-5 data layout for better scalability. MiPiL minimizes data migration while maintaining a uniform data distribution, not only for regular data but also for parity data.

2.1 Design Objectives

For RAID-5 scaling, it is desirable to ensure an even load on all the disks and to require minimal data migration. A parity block and a regular data block sustain loads with different access patterns. Therefore, not only should regular data be distributed evenly, parity data should also be distributed evenly. 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.

Assume that the ith RAID-5 scaling operation is performed from N_{i-1} disks to N_i, and that each disk consists of s data blocks. The following four requirements should be satisfied for RAID-5 scaling.

- Requirement 1 (Uniform Data and Parity Distributions): After this scaling operation, each one of the N_i disks holds $((N_{i-1}-1)\times s)/N_i$ data blocks and s/N_i parity blocks.
- Requirement 2 (Minimal Data Migration): During this scaling operation, the expected number of blocks to be moved is $(N_{i-1}\times s)\times(N_i-N_{i-1})/N_i$.
- Requirement 3 (Fast Data Addressing): After this scaling operation, the location of a block is computed by an algorithm with low time and space complexities for regular data and parity data.

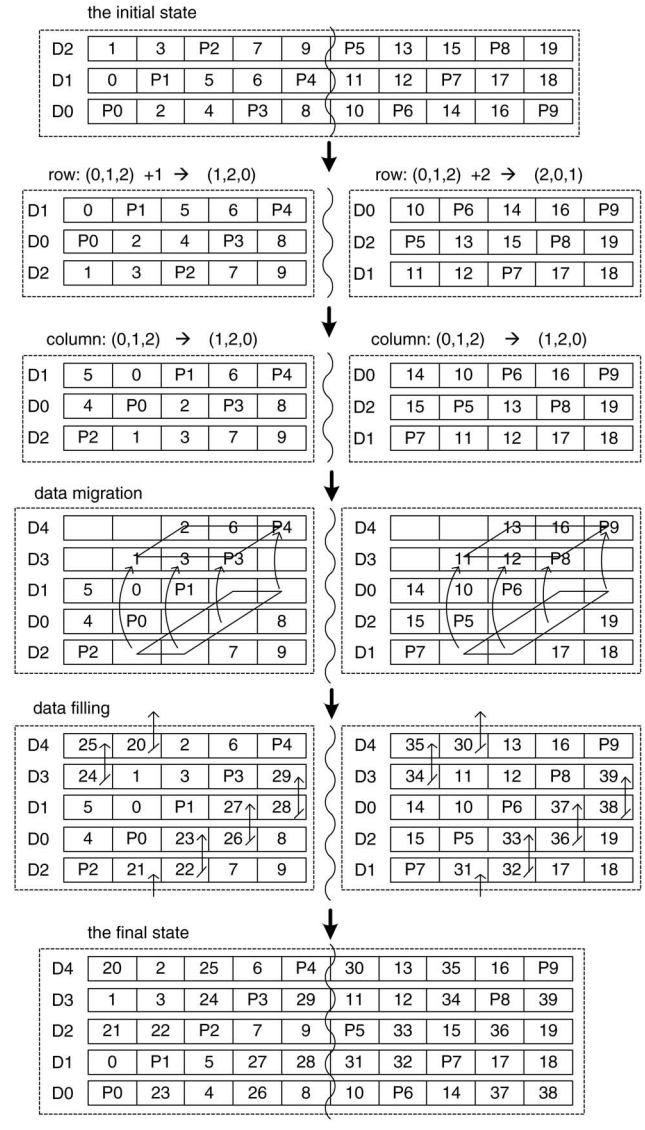

Fig. 1. RAID-5 scaling from 3 disks to 5 using MiPiL.

- Requirement 4 (Data Reliability during Scaling): Data are not lost or corrupted if the system or one disk fails in the middle of scaling.

2.2 Overview of How MiPiL Works

During a RAID-5 scaling, MiPiL 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. A RAID-5 scaling is over when data migration is finished. After that, newly added capacity is available, and new data can be filled into the RAID-5 gradually.

2.2.1 First Scaling Operation

To understand how the MiPiL approach works, we examine the first RAID-5 scaling operation from N_0 disks to N_1, where $N_0 < N_1$. As shown in Fig. 1, every N_1 consecutive locations in one disk are grouped into one *segment*. N_1 segments with the same physical address are grouped into one *region*. In Fig. 1, different regions are separated by wavy lines.

Before data migration, MiPiL introduces a normalizing operation to shuffle rows and columns, which helps to align parity blocks on the edge of migration parallelograms. This normalizing operation includes two steps. First, the row number is transformed by the equation $r = (r_0 + \delta) \bmod N_0$, where r_0 is the original row number, and δ is the difference between $N_0 - 1$ and the row number of the parity data in the final column. After this step, the parity blocks in the last N_0 columns are distributed in the minor diagonal. Second, the first N_0 columns are transformed with regard to the row number of the parity data in each column. Take Fig. 1 as an example, parity blocks P_0, P_1, and P_2 are in Columns 0, 1, and 2 respectively before the column transformation. After the column transformation, the three parity blocks are in Columns 1, 2, and 0 respectively. After this step, the parity blocks in the first N_0 columns are distributed in the minor diagonal. It should be noted that the normalization of the logical view does not result in any data migration.

Afterwards, as far as a region is concerned, all the data blocks within a parallelogram will be moved. The base of the parallelogram is $N_1 - N_0$, and the height is N_0. In other words, $N_1 - N_0$ data blocks are selected from each old disk and migrated to new disks. The $N_1 - N_0$ blocks are consecutive in the logical view. Fig. 1 depicts the moving trace of each migrating block. For one moving data block, only its row number is changed while its column number is unchanged.

After data migration, each disk, either old or new, has $N_0 - 1$ regular data blocks and one parity block within one region. This indicates that MiPiL regains a uniform data distribution. The total number of data blocks to be moved is the area of the moving parallelogram, i.e., $(N_1 - N_0) \times N_0$. This reaches the minimal number of moved blocks, $(N_0 \times N_1) \times (N_1 - N_0)/N_1 = (N_1 - N_0) \times N_0$. We can claim that the RAID-5 scaling using MiPiL can satisfy Requirement 1 and Requirement 2.

MiPiL moves a block by reading it and writing it into another location within the same stripe. This copying operation changes the total content of the stripe, and therefore requires a parity update. Subsection 3.1 depicts how MiPiL recomputes and updates the parity efficiently.

After a RAID-5 scaling, new data can be filled gradually. As shown in the final state in Fig. 1, $N_1 - N_0$ new data blocks are placed into each stripe sequentially. These new blocks are distributed in a round-robin order in the logical view (data filling in Fig. 1). See the description of the Placing( ) procedure in Subsection 2.3 for more details.

2.2.2 Successive Scaling Operations

From the last subsection, we conclude that MiPiL succeeds in meeting the first two requirements for the first scaling operation. In that scenario, the initial state is the standard RAID-5 layout, where parity information is distributed across all the old disks in a round-robin manner. For successive scaling operations, how MiPiL constructs a logical view where parity data are distributed in a round-robin manner is a grand challenge.

MiPiL uses two functions, $P_i(x)$ and $L_i(x)$, to help construct this logical view. $P_i(x)$ denotes the ordinal number of the disk holding the parity data in column x. For example, since Disk 2 holds the parity data for column 5, $P_0(5) = 2$.

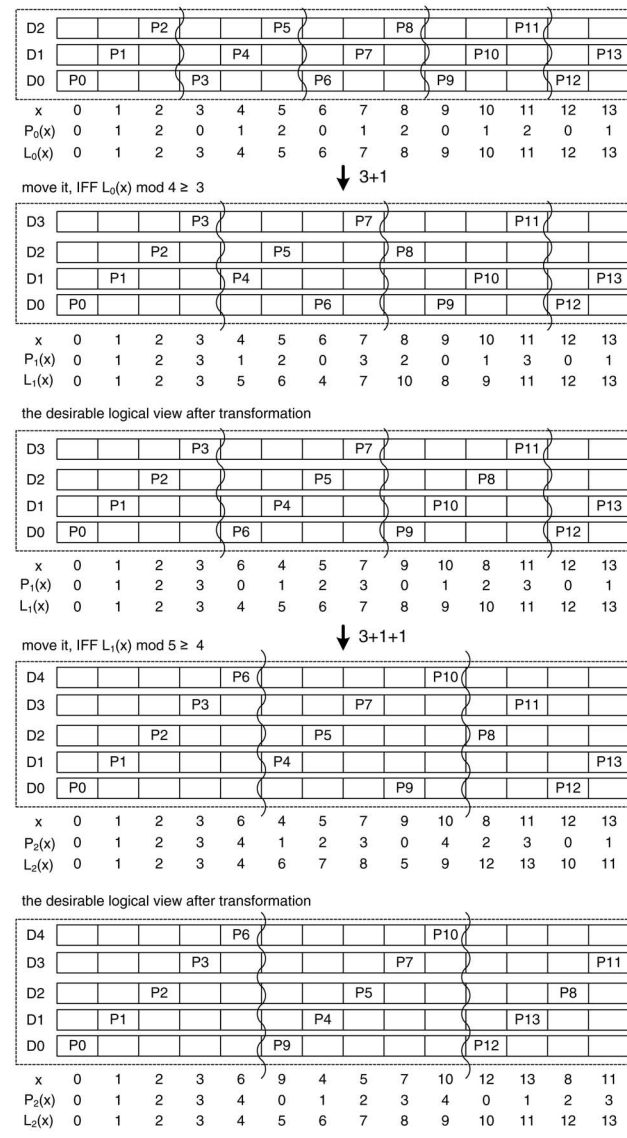

Fig. 2. MiPiL maintains the round-robin layout of parity data after multiple scaling operations.

$L_i(x)$ denotes the value for column x when counting columns. The counting order is sequential in the granularity of regions. Within a region, counting is in the order of $P_i(x)$. The following equation defines $P_i(x)$:

$$P_i(x) = \begin{cases} Q_i(x), & \text{if } Q_i(x) \geq N_{i-1}; \\ P_{i-1}(x), & \text{otherwise.} \end{cases} \quad (1)$$

The following equation defines $L_i(x)$:

$$L_i(x) = L_{i-1}(x) - Q_i(x) + P_i(x), \quad (2)$$

where $Q_i(x) = L_{i-1}(x) \bmod N_i$, and N_i gives the total number of disks after the ith scaling operation. Moreover, we have $L_0(x) = x$, and $P_0(x) = x \bmod N_0$. If $Q_i(x) \geq N_{i-1}$, the parity data is moved; otherwise, the parity data keep unmoved. This manner guarantees that the distribution of parity data is uniform and the migration of parity data is minimal.

Fig. 2 shows how MiPiL maintains the round-robin layout of parity data in the logical view after multiple scaling operations. For RAID-5 scaling from 3 disks to 4, only one parity

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

Fig. 3. The addressing algorithm in MiPiL.

block is moved for each region. To construct a desirable logical view, the four columns in each region are reorganized with regard to the value of $P_1(x)$. We can see that parity data is distributed in a round-robin manner.

Let us continue to examine a second scaling operation "$3 + 1 + 1$". For region 0, parity data P_6 is moved from disk 0 to disk 4. However, the other four parity blocks in region 0 are not moved. Likewise, only P_{10} is moved for region 1. After the five columns in each region are reorganized with regard to the value of $P_2(x)$, parity data is distributed in a round-robin manner. This desirable logical view will be the initial state for the next scaling operation.

We can see that after each scaling operation, the distribution of parity blocks in each region is even. Furthermore, the number of parity blocks to be moved is minimal. It should also be noted that the transformation of the logical view does not result in any data migration.

2.3 The Addressing Method

2.3.1 The Addressing Algorithm

Fig. 3 shows the addressing algorithm to minimize data migration required by RAID-5 scaling. The array N records the history of RAID-5 scaling. When a RAID is constructed from scratch (i.e., $t = 0$), it is a standard RAID-5 array made up of $N[0]$ disks actually. The address of block x can be calculated via one division and one modular (lines 3-6). After

Algorithm: Normalizing(m, n, d, b, r, c).

Input: The input parameters are m, n, d, and b, where
m: the number of original disks;
n: the number of new disks;
d: the disk number;
b: the physical block number.
Output: The output data are r and c, where
r: the target row number;
c: the target column number.

```
(1) L ← L_{t−1}(b) ;
(2) lastC ← L − (L mod (m + n)) + (m + n − 1);
(3) lastP ← lastC mod m;
(4) δ ← m − 1 − lastP; //the distance to move
(5) r ← (d + δ) mod m;
(6) c ← L mod (m + n);
(7) if (c < m) then
(8)     c ← ((L mod m) + δ) mod m.
```

Fig. 4. The normalizing procedure.

the ith scaling operation, the RAID-5 array consists of $N[i]$ disks.

Let us examine the tth scaling operation, where n disks are added into a RAID made up of m disks (lines 8-9). First, if block x is an original block (line 10), MiPiL calculates its old address (d_0, b_0) before the tth scaling (line 11). Then, block (d_0, b_0) is normalized into point (r, c) in the logical view (line 12). If (r, c) needs to be moved, MiPiL changes the disk number (line 15). Otherwise, MiPiL keeps the disk number unchanged (line 17). Second, if block x is a new block, MiPiL places it via the placing procedure (line 19).

The code of line 14 is used to decide whether data block x will be moved. As shown in Fig. 1, there is a parallelogram in each region. The base of the parallelogram is n, and the height is m. A data block will be moved if and only if it is within a parallelogram. One parallelogram mapped to row r is a line segment. Its beginning and ending columns are $r + 1$ and $r + n$, respectively. If c is within the line segment, block x is within the parallelogram, and therefore it will be moved.

2.3.2 The Normalizing Procedure

The goal of normalizing (see Fig. 4) is to make the logical view of the data layout regular so as to simplify subsequent calculations. MiPiL locates the last column in the current region (line 2), and the row number of the parity block in the last column (line 3). First, to make the parity data in the last column onto row $m - 1$, the row number is transformed by the equation $r = (d + \delta) \bmod m$ (line 5). Second, the first m columns are reorganized with regard to the row number of the parity data in each column (lines 7-8). It should be noted that the normalization of the logical view does not result in data migration.

2.3.3 The Moving Function

Once a data block is determined to be moved, MiPiL changes its disk number with the moving function (see Fig. 5). As shown in Fig. 6, a migrating parallelogram is divided into three parts, i.e., a head triangle, a body parallelogram, and a

Algorithm: Moving(m, n, r, c).

Input: The input parameters are m, n, r, and c, where
m: the number of original disks;
n: the number of new disks;
r: the row number after normalizing;
c: the column number after normalizing.
Output: The disk number after a block is moved.

```
if m ≥ n then                                              (1)
    if c < n then                                          (2)
        return r + m;                                      (3)
    if c > m then                                          (4)
        return r + n;                                      (5)
    return m + n − (c − r);                                (6)
else                                                       (7)
    if c < m then                                          (8)
        return r + m;                                      (9)
    if c > n then                                          (10)
        return r + n;                                      (11)
    d ← r + c;                                             (12)
    if r < m − 1 − (m + n − 1 − c) mod m then              (13)
        d ← d + 1;                                         (14)
    return d.                                              (15)
```

Fig. 5. The moving function.

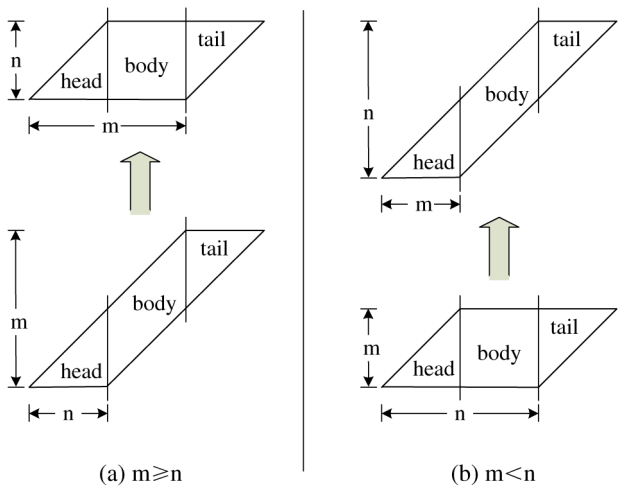

Fig. 6. The variation of data layout involved in migration.

tail triangle. How a 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 rows (lines 3, 9), while the tail triangle will be moved by n rows (lines 5, 11). The body is twisted from a parallelogram to a rectangle when $m \geq n$ (line 6), while from a rectangle to a parallelogram when $m < n$ (line 12). MiPiL keeps the relative locations of all blocks in the same column. The only exception is for a body parallelogram when $m < n$. If a regular data block has a smaller row number than the parity block in the same column (line 13), this data block will move upwards by one more row (line 14). The goal is to guarantee the uniform distribution of

Algorithm: Placing(m, n, s, x, d, b).

Input: The input parameters are m, n, s, and x, where
m: the number of original disks;
n: the number of new disks;
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.

```
y ← x − (m − 1) × s;                                       (1)
w ← y mod n;                                               (2)
b ← y/n;                                                   (3)
Normalizing(m, n, 0, b, δ, c);                             (4)
r ← (m + c + w) mod (m + n);                               (5)
if (r < m) then                                            (6)
    d ← (r + m − δ) mod m;                                 (7)
else                                                       (8)
    d ← r.                                                 (9)
```

Fig. 7. The placing procedure.

parity blocks. In this case, the relative locations between the regular data and the parity data are not maintained.

2.3.4 The Placing Procedure

When block x is at a location newly added after the last scaling, it is addressed via the placing procedure (see Fig. 7). Block x is the yth new block (line 1). Each stripe holds n new blocks. So, we have $b = y/n$ (line 3). In turn, MiPiL normalizes b to get the corresponding column number c in the logical view (line 4). As shown in Fig. 1, the order of placing new blocks in each column in the logical view is obvious and simple (see the upward arrows). In the logical view, MiPiL gets the row number r according to the column number c (line 5). If $r < m$, MiPiL denormalizes r to get the disk number d (line 7); otherwise, the disk number d is just the row number r (line 9).

The addressing algorithm used in MiPiL is very simple. It requires fewer than 100 lines of C code, reducing the likelihood that a bug will cause a data block to be mapped to the wrong location.

2.4 Satisfaction of the Requirements

The purpose of this experiment is to quantitatively characterize whether the MiPiL approach satisfies the first three requirements described in Subsection 2.1. How MiPiL satisfies Requirement 4 will be discussed in Section 3. For this purpose, we compare MiPiL with the round-robin approach. From a four-disk array, we add one disk repeatedly for 10 times using the two approaches respectively. Each disk has a capacity of 128 GB, and the size of a data block is 64 KB. In other words, each disk holds 2×1024^2 blocks.

Uniform data distribution. We use the coefficient of variation as a metric to evaluate the uniformity of data distribution across all the disks. The coefficient of variation expresses the standard deviation as a percentage of the average. For the two algorithms, the coefficients of variation remain 0% as the times of disk additions increases. This indicates that MiPiL

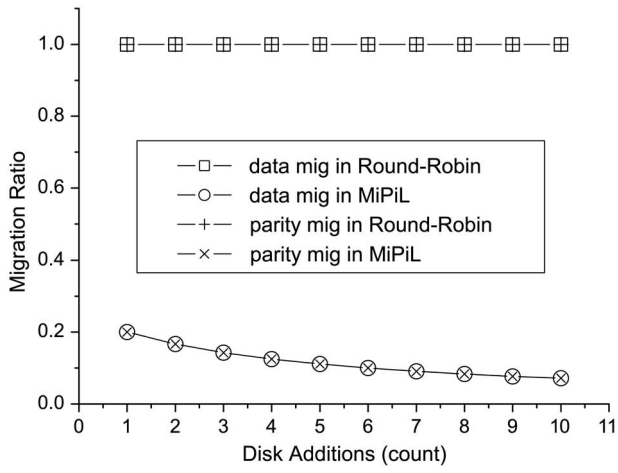

Fig. 8. Comparison of data migration ratio.

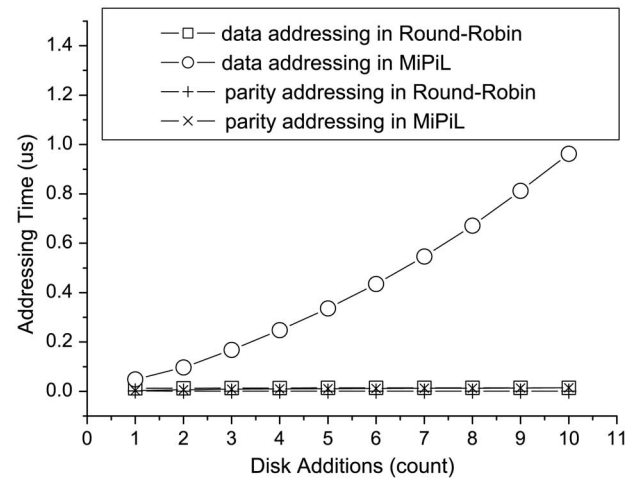

Fig. 9. Comparison of addressing time.

succeeds in maintaining uniform distributions of regular data and parity data.

Minimal Data Migration: Fig. 8 plots the migration fraction (i.e., the fraction of data blocks to be migrated) versus the number of scaling operations. Using the round-robin approach, the migration fraction is constantly 100%. This "moving-everything" approach will cause very expensive migration cost.

The migration fractions of regular data and parity data using MiPiL are significantly smaller than the migration fractions using the round-robin approach. To gain a uniform data distribution, the minimal fractions of data blocks and parity blocks to be moved for RAID-5 scaling are identical to the percentage of new disks. Our calculation indicates that the migration fractions with MiPiL reach this theoretical minimum. Another obvious phenomenon is that the migration fractions decrease with the increase of the number of scaling operations. The reason behind this phenomenon is that the migration fractions with MiPiL are identical to the percentage of new disks, which decreases with the number of scaling operations.

Fast Data Addressing: We run different algorithms to calculate the physical addresses for all data blocks on a scaled RAID. The whole addressing process is timed and then the average addressing time for each block is calculated. The testbed used in the experiment is an Intel Dual Core i7-2640M 2.80 GHz machine with 4 GB of memory. A Windows 7 Enterprise Edition is installed.

Fig. 9 plots the addressing time versus the number of scaling operations. The round-robin algorithm has a low calculation overhead in data addressing and in parity addressing. MiPiL also has a low calculation overhead in parity addressing. The calculation overhead using MiPiL in data addressing increases gradually. Fortunately, the largest addressing time using MiPiL is 0.96 μs which is negligible compared to milliseconds of disk I/O time.

The reason that data addressing time of the MiPiL approach is worse than that of the counterpart approach is demonstrated as follows. Let $T(t)$ denote the time complexity of the Addressing algorithm. Since Addressing is a recursive algorithm, we can represent $T(t)$ by using a recurrence relation. First, it is clear that $T(0) = c_1$ for some constant c_1. Next, we notice that both the Moving function and the Placing procedure take constant time. Thus, we have $T(t) \leq T(t-1) + c_2$, for all $t \geq 1$, where c_2 is some constant. Solving the above recurrence relation, we get $T(t) \leq c_1 + c_2 t = O(t)$. The Addressing algorithm of MiPiL has time complexity $O(t)$ after t RAID scalings.

3 Optimization Techniques

The MiPiL approach succeeds in minimizing data migration for RAID-5 scaling. In this section, we describe how MiPiL optimizes the process of data migration. Meanwhile, we can see how MiPiL satisfies Requirement 4 described in Subsection 2.1.

3.1 Piggyback Parity Updates

Data reliability is ensured by RAID-5 by maintaining the parity information as the XOR sum of all the data blocks in a stripe. MiPiL copies some blocks in a stripe, without need of erasing old blocks. This changes the total content of the stripe, and therefore requires a parity update. MiPiL piggybacks parity updates during data migration to minimize the numbers of additional XOR computations and disk I/Os.

Before performing data migration for scaling, new disks are zeroized. This zeroizing operation does not take up the scaling time. To consider how one parity block is updated, we divide data stripes in the RAID into four categories.

In a stripe of the first category, no data migration is required. As shown in Fig. 10, stripe 2 is in this category. MiPiL does not change any content of the stripe, and therefore does not require a parity update. In this case, no additional XOR computation or disk I/O is needed.

In a stripe of the second category, regular data blocks are migrated and no parity block is moved. As shown in Fig. 10, stripes 0 and 1 are in this category. Without loss of generality, we assume that blocks in a stripe are B_0, B_1, B_2, B_3, and P before scaling. After data migration, blocks in this stripe are B_0, B_1, B_2, B_3, Q, B_2, and B_3. Since the original parity block $P = B_0 \oplus B_1 \oplus B_2 \oplus B_3$, we have the new parity block $Q = B_0 \oplus B_1 \oplus B_2 \oplus B_3 \oplus B_2 \oplus B_3 = P \oplus B_2 \oplus B_3$. Since B_2 and B_3 need to be moved, they will be in memory and available for computing parity. To update parity, only a parity

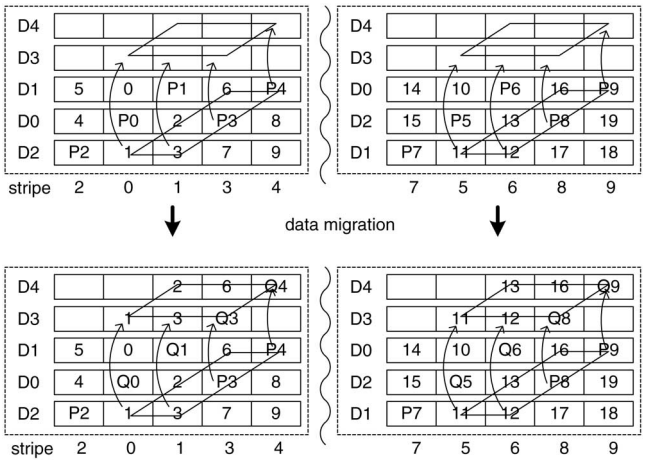

Fig. 10. Parity changes across data migration. Copying some blocks within a stripe changes the total content of the stripe, and therefore requires a parity update.

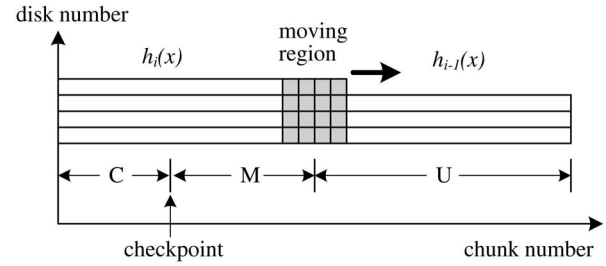

Fig. 11. Lazy updates of mapping metadata. "C": migrated and checkpointed; "M": migrated but not checkpointed; "U": not migrated. Data redistribution is checkpointed only when a user write request arrives in the area "M".

read and a parity write are added. We take Stripe 0 as an example, only P_0 is read and Q_0 is written.

In a stripe of the third category, regular data blocks are migrated and a parity block is moved. As shown in Fig. 10, stripe 3 is in this category. Without loss of generality, we assume that blocks in a stripe are B_0, B_1, B_2, B_3, and P before scaling. After data migration, blocks in this stripe are B_0, B_1, B_2, B_3, P, B_2, B_3, and Q. Since the original parity block $P = B_0 \oplus B_1 \oplus B_2 \oplus B_3$, we have the new parity block $Q = B_0 \oplus B_1 \oplus B_2 \oplus B_3 \oplus P \oplus B_2 \oplus B_3 = P \oplus P \oplus B_2 \oplus B_3 = B_2 \oplus B_3$. Since B_2 and B_3 need to be moved, they will be in memory and available for computing parity. By consideration of updating parity, a block read is eliminated and no additional write is added.

In a stripe of the fourth category, no regular data block is migrated and a parity block is moved. As shown in Fig. 10, stripe 4 is in this category. Without loss of generality, we assume that blocks in a stripe are B_0, B_1, B_2, B_3, and P before scaling. After data migration, blocks in this stripe are B_0, B_1, B_2, B_3, P, and Q. Since $P = B_0 \oplus B_1 \oplus B_2 \oplus B_3$, we have $Q = B_0 \oplus B_1 \oplus B_2 \oplus B_3 \oplus P = P \oplus P = 0$. Since the content of newly added disks is 0, a block read and a block write are eliminated by consideration of updating parity. We take Stripe 4 as an example, there is no need to write Q_4.

As far as a region during a RAID-5 scaling from m disks to $m+n$ is concerned, the numbers of stripes in the four categories are respectively 1, $m-1$, $n-1$, and 1. By consideration of updating parity, $1 \times 0 + (m-1) \times 1 + (n-1) \times (-1) + 1 \times (-1) = m - n - 1$ additional reads are required, and $1 \times 0 + (m-1) \times 1 + (n-1) \times 0 + 1 \times (-1) = m - 2$ additional writes are required. A region includes $m+n$ stripes in total. Assume that each disk consists of s data blocks. On account of updating parity, $s \times (m-n-1)/(m+n)$ additional blocks are read, and $s \times (m-2)/(m+n)$ additional blocks are written, so as to complete all the data migration for a RAID-5 scaling. Supposing a three disk RAID-5 is scaled to five disks, we have $m = 3$ and $n = 2$. On account of updating parity, no additional block is read, and $20\% \times s$ additional blocks are written.

We can see that MiPiL updates parity data without delay. When a disk failure happens during a RAID-5 scaling, lost data can be recovered from surviving disks. See Section 3.2 for more details about how the system knows which disks to include in parity calculations to make this possible.

3.2 Lazy Metadata Updates

While data migration is in progress, the RAID storage serves user requests. Furthermore, the incoming user I/Os may be write requests to migrated data. If mapping metadata do not get updated until all of the blocks have been moved, data consistency may be destroyed. Ordered operations [12] of copying a data block and updating the mapping metadata (a.k.a., *checkpoint*) can ensure data consistency. However, ordered operations require frequent metadata writes. Because metadata are usually stored at the beginning of all member disks, each metadata update causes one long seek per disk, which increases the cost of data migration significantly. MiPiL uses lazy metadata updates to minimize the number of metadata writes without compromising data reliability.

The key idea behind lazy metadata updates is that data blocks are copied to new locations and parity blocks are updated continuously, while mapping metadata are not updated onto the disks until a threat to data consistency appears. We use $h_i(x)$ to describe the geometry after the ith scaling operation, where N_i disks serve user requests. Fig. 11 illustrates an overview of the migration process. Data in the moving region are 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 data migration within the moving region is finished, the next region becomes the moving region. In this way, the newly added disks are gradually available to serve user requests. Only when a user write request arrives in the area where data have been moved and the movement has not been checkpointed, are mapping metadata updated.

The foundation of lazy metadata updates is described as follows. MiPiL only moves data blocks from old disks to new disks, which will not overwrite any valid data. After a data block is copied, both its new and original replicas are valid. As shown in Fig. 12, we suppose that blocks 1, 2, 3, 6, P_3, and P_4 have been copied to their new locations and the mapping metadata have not been updated, when the system fails. The original replicas of blocks 1, 2, 3, and 6 will be used after the system reboots. As long as the four data blocks have not been written since they are copied, the data remain consistent. Generally speaking, when the mapping information is not updated immediately after a data block is copied, a crash or

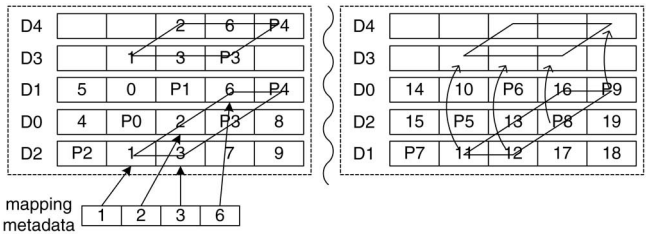

Fig. 12. Data blocks are copied to their new locations and metadata are not yet updated when the system fails. In this case, data consistency is still maintained because the data in their original locations are valid and available.

power outage does not result in data loss. The only threat to data consistency is the incoming of write operations to migrated data.

One important concern is whether data will be lost if a member disk in a RAID-5 array fails after a power failure. As far as a stripe group in the "M" or "U" area in Fig. 11 is concerned, MiPiL recovers lost data in the following manner.

- If there is no block to be moved, MiPiL recovers lost data in the old stripe group with N_{i-1} disks. In this case, the original parity is the XOR sum of $N_{i-1}-1$ original data blocks.
- If there are blocks to be moved and these blocks do not include a parity block, MiPiL recovers lost data in the new stripe group with N_i disks. If data migration is performed, the parity block is updated into the XOR sum of N_i-1 data blocks. Otherwise, since the content of newly added disks is 0, the parity block can also be considered as the XOR sum of N_i-1 data blocks.
- If there are blocks to be moved and these blocks include a parity block, MiPiL recovers lost data in the old stripe group with N_{i-1} disks. In this case, the original parity is not updated. Therefore, it is the XOR sum of $N_{i-1}-1$ data blocks.

Since one write of metadata can store multiple map changes of data blocks, lazy updates can minimize the number of metadata writes and reduce the cost of data migration significantly. Furthermore, lazy metadata updates can guarantee data consistency and keep the competence of protection against a single disk failure. Even if the system fails unexpectedly, only some data migrations are wasted. It should also be noted that the probability of a system failure is very low.

4 Implementation

We implement MiPiL 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-5 volumes [10]. Implementing MiPiL in MD makes it convenient to make a performance comparison between MiPiL and MD-Reshape. About 700 lines of code, counted by the number of semicolons and braces, are modified or added to the MD driver.

According to the addressing algorithm, MD forwards incoming I/O requests to corresponding disks. When a RAID-5 scaling begins, MD creates a kernel thread to perform data redistribution. MiPiL cannot redistribute a new region until all the I/O requests already issued to this region are completed.

MiPiL uses three variants to track how the expansion is progressing and determine the movement stage that the target part of the volume involves (see Fig. 11). While a region is being redistributed, any I/O attempt into the region is blocked until the redistribution of this region is finished.

When a user I/O arrives, MiPiL detects whether this is a write request, and it arrives in the area where data have been moved and the movement has not been checkpointed. If so, mapping metadata are updated before the write request is served.

5 Experimental Evaluation

This section presents results of a comprehensive experimental evaluation comparing MiPiL with MD-Reshape. MD-Reshape preserves a round-robin data distribution after adding disks. Therefore, it moves 100 percent of data blocks.

5.1 Evaluation Methodology

We evaluate our design by running trace-driven experiments over a real system. 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.

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

- Financial1 is obtained from the Storage Performance Council (SPC) [13], [14], a vendor-neutral standards body. The Financial1 trace was collected from OLTP applications running at a large financial institution. The write ratio is high.
- TPC-C traced disk accesses of the TPC-C database benchmark with 20 warehouses [15]. 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.

To replay I/O traces, we implement a block-level replay tool using Linux kernel 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.

5.2 Experiment Results

We evaluate the MiPiL approach in terms of its performance during scaling and its performance after scaling.

5.2.1 Performance During Scaling

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 MiPiL through a comparison with MD-Reshape. We conduct a scaling operation of adding one disk to a four-disk RAID, where each disk has a capacity

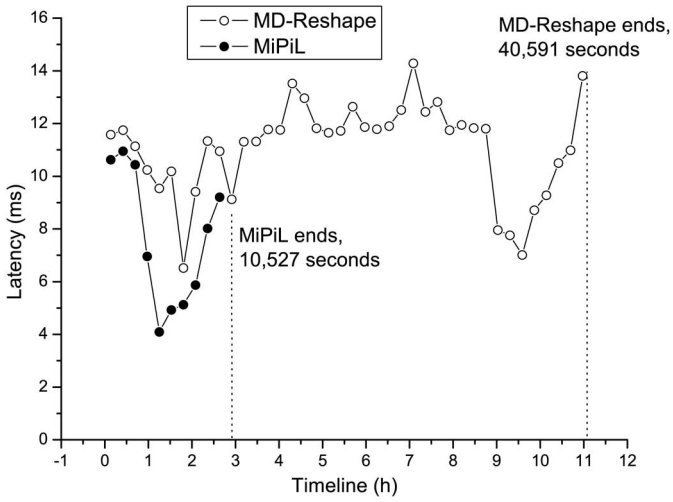

Fig. 13. Performance comparison between MiPiL and MD-Reshape for the Financial1 workload.

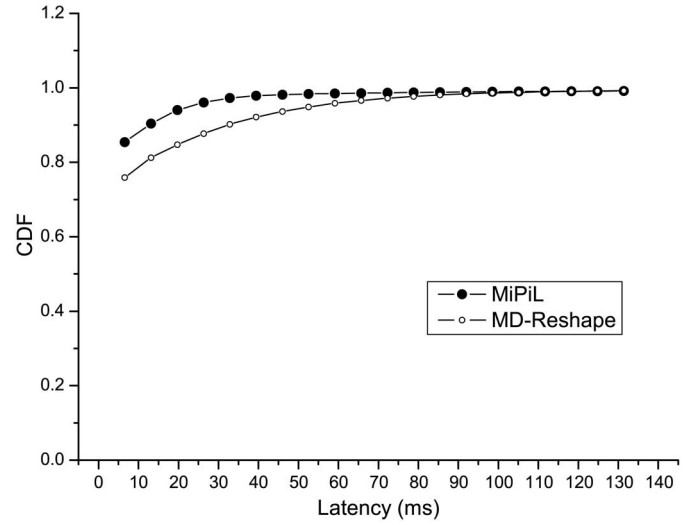

Fig. 14. Cumulative distribution of I/O latency during data redistribution by the two approaches for the Financial1 workload.

of 100 GB. Each approach performs with the 256KB chunk size under a Financial1 workload.

During a RAID-5 scaling, data redistribution and foreground applications share and even contend for I/O resources in the system. Generally speaking, the faster the redistribution performs, the worse the applications performance is. 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 chose control parameters for different experiments. The parameters of "sync_speed_max"and "sync_speed_min" in MD-Reshape are set as 200,000 and 2,000 respectively. In MiPiL, they are set as 200,000 and 10,000 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 1000 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. 13 plots local average latencies using the two approaches as the time increases along the x-axis. It illustrates that MiPiL demonstrates a noticeable improvement over MD-Reshape in two metrics.

First, the redistribution time using MiPiL is significantly shorter than that using MD-Reshape. They are 10,527 seconds and 40,591 seconds, respectively. In other words, MiPiL has a 74.07% shorter redistribution time than MD-Reshape.

The main factor in MiPiL's reducing the redistribution time is the significant decline of the amount of the data to be moved. When MD-Reshape is used, 100% of data blocks have to be migrated. However, when MiPiL is used, only 20% of data blocks need to be migrated. Another factor is the effective exploitation of two optimization techniques, i.e., piggyback parity updates to reduce additional XOR computations and disk I/Os, and lazy metadata updates to minimize metadata writes.

Second, local average latencies of MD-Reshape are obviously longer than those of MiPiL. The global average latency using MD-Reshape reaches 10.26 ms while that using MiPiL is 7.62 ms. In other words, MiPiL brings an improvement of 25.78% in user response time. Fig. 14 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 10,527 seconds. For any I/O latency smaller than 78.83 ms, the CD value of MiPiL is greater than that of MD-Reshape noticeably and consistently. This indicates again that MiPiL has smaller response time of user I/Os than MD-Reshape.

The reason for the improvement in user response time is explained as follows. During a RAID-5 scaling, data redistribution and foreground applications share and even contend for I/O resources in the system. MiPiL decreases the amount of the data to be moved significantly. Moreover, MiPiL minimizes metadata writes via lazy metadata updates. As a result, the RAID system has more time to serve applications. It is also noteworthy to mention that due to significantly shorter data redistribution time, MiPiL has a markedly lighter impact on the user I/O latencies than MD-Reshape does.

A factor that might affect the benefits of MiPiL is the type of workload under which data redistribution performs. Under the TPC-C workload, we also measure the performance of MiPiL and MD-Reshape in performing the "$4+1$" scaling operation.

For the TPC-C workload, Fig. 15 shows local average latency versus the redistribution time for MD-Reshape and MiPiL. It shows once again the efficiency of MiPiL in improving the redistribution time. The redistribution times using MD-Reshape and MiPiL are 13.02 hours and 2.92 hours, respectively. That is to say, MiPiL has an improvement of 77.57% in the redistribution time. Likewise, local average latencies of MiPiL are also obviously shorter than those of MD-Reshape. The global average latency using MiPiL is 0.41 ms while using MD-Reshape reaches 1.39 ms. In other words, MiPiL has an improvement of 70.50% in user response time.

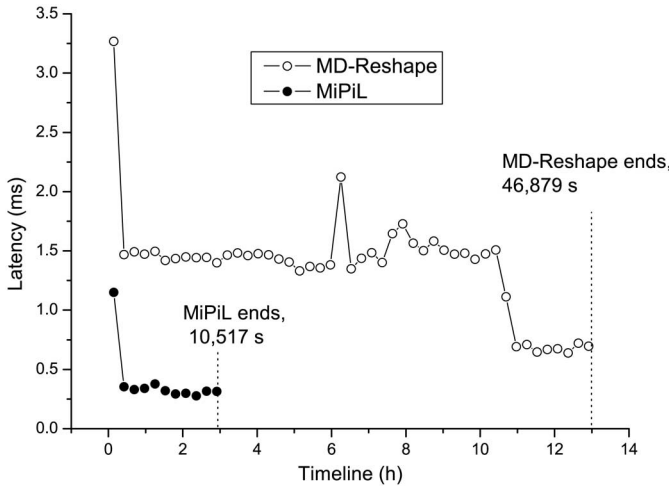

Fig. 15. Performance comparison between MiPiL and MD-Reshape for the TPC-C workload.

We can draw one conclusion from the above two experiments. Under various workloads, MiPiL can consistently outperform MD-Reshape by 74.07-77.57% in redistribution time and by 25.78-70.50% in user response time.

5.2.2 Performance after Scaling

The above experiments show that MiPiL improves the scaling efficiency of RAID-5 significantly. One of our concerns is whether there is a penalty in the performance of the data layout after scaling using MiPiL, compared with the round-robin layout preserved by MD-Reshape.

We use the WebSearch2 and Financial1 workloads to measure the performance of the two RAIDs, scaled from the same RAID using MiPiL 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. 16 plots the local average latency of the two RAIDs under the WebSearch2 workload as the time increases along the x-axis. We can find that the performance of the MiPiL RAID is better than that of the round-robin RAID. With regard to the round-robin RAID, the average latency is 1.05 ms. For the MiPiL RAID, the average latency is 0.86 ms. That is to say, MiPiL improves the RAID-5 performance by 18.10%. Fig. 17 plots the local average latency of the two RAIDs under the Financial1 workload. With regard to the round-robin RAID, the average latency is 3.18 ms. For the MiPiL RAID, the average latency is 3.07 ms. MiPiL improves the RAID-5 performance by 3.5%.

Second, we compare the performance of two RAIDs, after two scaling operations "$4+1+1$" using the two approaches. Fig. 18 plots local average latencies of the two RAIDs under the WebSearch2 workload as the time increases along the x-axis. It again reveals the difference in the performance of the two RAIDs. With regard to the round-robin RAID, the average latency is 1.01 ms. For the MiPiL RAID, the average latency is 0.86 ms. Fig. 19 plots the local average latency of the two RAIDs under the Financial1 workload. With regard to

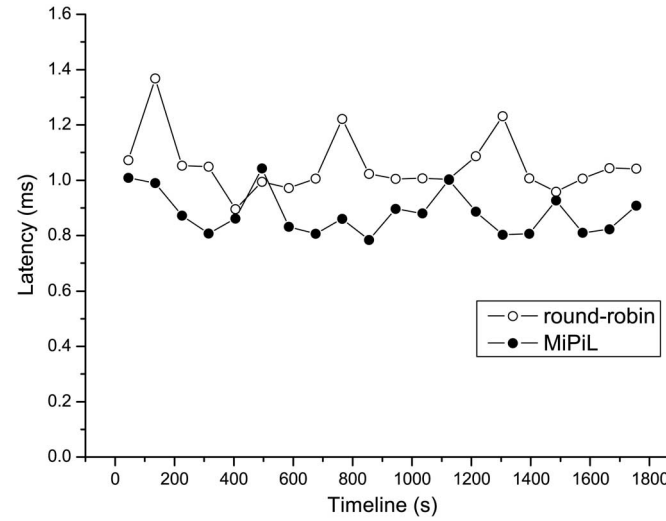

Fig. 16. Performance comparison between MiPiL layout and round-robin layout for the WebSearch2 workload after one scaling operation "$4+1$".

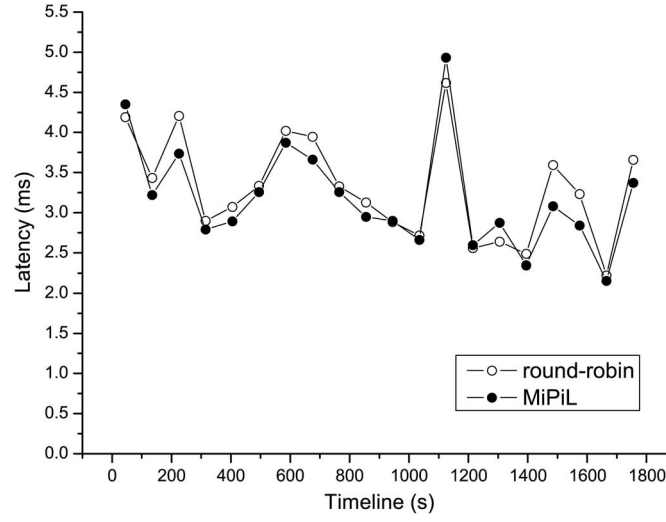

Fig. 17. Performance comparison between MiPiL layout and round-robin layout for the Financial1 workload after one scaling operation "$4+1$".

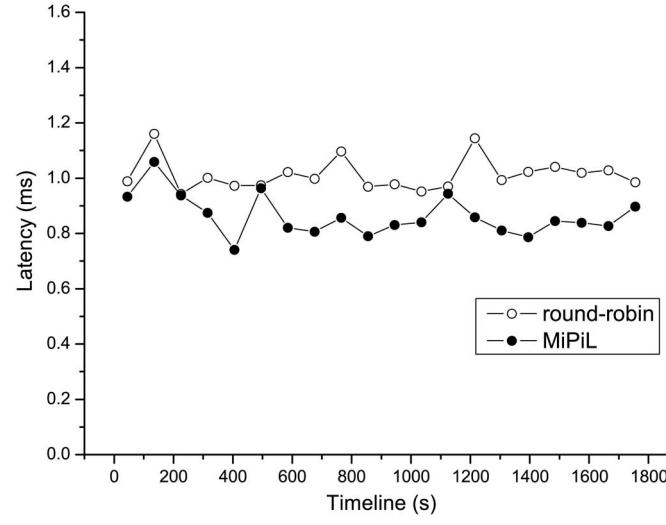

Fig. 18. Performance comparison between MiPiL layout and round-robin layout for the WebSearch2 workload after two scaling operations "$4+1+1$".

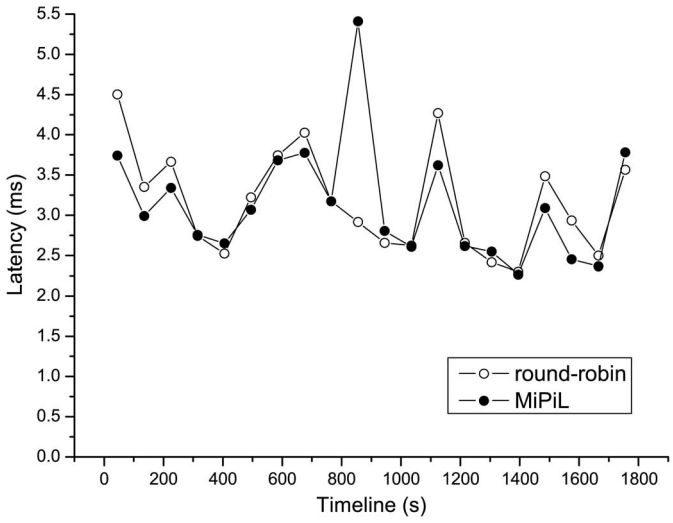

Fig. 19. Performance comparison between MiPiL layout and round-robin layout for the Financial1 workload after two scaling operations "$4+1+1$".

the round-robin RAID, the average latency is 3.06 ms. For the MiPiL RAID, the average latency is 3.08 ms. MiPiL lowers the RAID-5 performance by 0.7%.

Table 1 summarizes these performance differences between the MiPiL RAID and the round-robin RAID-5 after different scaling cases. We can reach the conclusion that under the WebSearch2 and Financial1 workloads, the performance of the RAID scaled using MiPiL is almost identical to that of the round-robin RAID-5.

TABLE 1
The Percentages by Which MiPiL Improves or Degrades the Performance after Different Scaling Cases (An Up Arrow Indicates an Performance Improvement, While a Down Arrow Indicates a Performance Degradation)

scaling \ workload	WebSearch2	Financial1
$4+1$	$18.1\% \uparrow$	$3.5\% \uparrow$
$4+1+1$	$14.85\% \uparrow$	$0.7\% \downarrow$

6 Related Work

In this section, we first review related work that addresses block-based disk scaling. There are many efforts on randomized data distribution, which has the ability of disk scaling. Some people insist that it is unnecessary to address block-level disk scaling. To argue against this viewpoint, we then review this category of work, and claim that these randomized strategies are designed for object-based storage systems, instead of block-level storage devices. There is also a concern that the scaling problem can be solved more easily at a higher level (e.g., file system). Consequently, we finally discuss RAID scaling with the support of file systems.

6.1 RAID Scaling for Block-Based Disks

The HP AutoRAID [16] allows an online capacity expansion without requiring data migration. Newly created RAID-5 volumes use all the disks in the system, but previously created RAID-5 volumes continue to use only the original disks.

The conventional approaches to RAID-5 scaling redistribute data and preserve the round-robin order after adding disks. Gonzalez and Cortes [9] proposed a gradual assimilation approach (GA) to control the speed of a RAID-5 scaling. Brown designed a reshape toolkit in a Linux MD driver (MD-Reshape) [10]. It writes mapping 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 frequent.

Zhang et al. [11], [17] 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.

A patent [18] presented 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.

The MDM method [19] eliminates the parity modification cost of a 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. It does also not increase (just maintains) the data storage efficiency after adding more disks.

Zheng et al. [20] proposed the FastScale approach to RAID-0 scaling. FastScale minimizes data migration while maintaining a uniform data distribution. FastScale does not handle RAID-5 scaling. In RAID-5, a parity block sustains a different load from what a regular block does. As a result, RAID-5 scaling is more challenging.

Zhao et al. [8] proposed an approach generating the encoding matrices for distributed storage systems based on E-MSR codes. This approach minimizes the changes of encoded blocks when scaling.

Wu et al. [21] proposed the GSR approach to accelerate RAID-5 scaling. GSR divides data on the original array into two consecutive sections. During RAID-5 scaling, GSR moves the second section of data onto the new disks, while keeping the first section of data unmoved. In this way, GSR minimizes data migration and parity modification. The main limitation of GSR is the performance of RAID systems after scaling. Only the original disks serve accesses to the first section of data. Only the new disks serve accesses to the second section of data. Especially, there is a large performance penalty when the workload exhibits a strong locality in its access pattern.

Franklin et al. [22] 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. Similar to WorkOut [23], this kind of method requires spare disks to be available.

6.2 RAID Scaling for Object-Based Storage

With the development of object-based storage, randomized RAID [24]-[27] is now gaining the spotlight in the data placement area. The cut-and-paste placement strategy [27] uses randomized allocation strategy to place data across disks. For a disk addition, it cuts off some data from old

disks, and pastes them to the newly added disk. Seo and Zimmermann [28] proposed an approach finding a sequence of disk additions and removals for the disk replacement problem. The goal is to minimize the data migration cost. Both of these two approaches assume the existence of a high-quality hash function. However, they did not present such a hash function.

The SCADDAR algorithm [24] uses a pseudo-random function to minimize the amount of data to be moved. Unfortunately, the pseudo-random function does not preserve the randomness of the data layout after several disk additions or deletions [28].

RUSH [29], [30] and CRUSH [31] 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 [32] 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.

6.3 RAID Scaling with Filesystem Support

With the support of the ZFS file system, RAID-Z [33] achieves acceptable scalability in distributed storage systems. RAID-Z is a data/parity scheme like RAID-5, but it uses dynamic stripe width. There is no simple formula to identify a stripe. Traversing the filesystem metadata is required to determine the RAID-Z geometry. HDFS RAID [34] is another instance that depends on a file system to achieve acceptable scalability.

Different from these solutions, MiPiL provides a block-level solution for RAID-5 scaling without need of any file system semantics.

7 Conclusions and Future Work

This paper proposes a new approach to RAID-5 scaling called MiPiL by rethinking RAID-5 data layout. First, with a new and elastic addressing algorithm, MiPiL minimizes the number of data blocks to be migrated without compromising the uniformity of data distribution. Second, MiPiL optimizes online data migration with piggyback parity updates and lazy metadata updates.

Our results from detailed experiments using real-system workloads show that, compared with MD-Reshape, a scaling toolkit released in 2010, MiPiL can reduce redistribution time by up to 74.07-77.57% and reduce user response time by 25.78-70.50%. The experiments also illustrate that under the Web-Search2 and Financial1 workloads, the performance of the RAID-5 scaled using MiPiL is almost identical to that of the round-robin RAID-5. We believe that this paper has made new contribution to modern storage management and will inspire more and general research interest in the area of operating systems.

MiPiL does not handle RAID-6 scaling. With higher possibility of multiple disk failures [35], [36], RAID-6 has received more attention than ever. We believe that MiPiL provides a good starting point for efficient scaling of RAID-6 arrays. In the future, we will focus on RAID-6 scaling.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants 60903183, 61170008, and 61272055, the National High Technology Research and Development Program of China under Grant 2013AA01A210, and the National Grand Fundamental Research 973 Program of China under Grant No. 2014CB340402. The work of K. Li was partially conducted when he was visiting the National Laboratory for Information Science and Technology, Tsinghua University, in the summer of 2012, as an Intellectual Ventures endowed visiting chair professor.

References

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

[2] P. Chen, E. Lee et al., "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, vol. 26, no. 2, p. 145C185, Jun. 1994.

[3] X. Yu, B. Gum et al., "Trading Capacity for Performance in a Disk Array," *Proc. Operating Systems Design and Implementation (OSDI'00)*, pp. 243-258, Oct. 2000.

[4] S. Ghandeharizadeh and D. Kim, "On-Line Reorganization of Data in Scalable Continuous Media Servers," *Proc. 7th Int'l Conf. Database and Expert Systems Applications*, pp. 755-768, Sep. 1996.

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

[6] K. Hwang, H. Jin, and R. Ho, "RAID-x: A New Distributed Disk Array for I/O-Centric Cluster Computing," *Proc. High Performance Distributed Computing (HPDC'00)*, pp. 279-286, Aug. 2000.

[7] Y. Saito, S. Frolund et al., "FAB: Building Distributed Enterprise Disk Arrays from Commodity Components," *Proc. Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS'04)*, pp. 48-58, Oct. 2004.

[8] H. Zhao, Y. Xu, and L. Xiang, "Scaling Up of E-Msr Codes Based Distributed Storage Systems with Fixed Number of Redundancy Nodes," *Int'l J. Distributed and Parallel Systems (IJDPS)*, vol. 3, no. 5, pp. 1-12, Sep. 2012.

[9] 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*, pp. 17-24, Sep. 2004.

[10] N. Brown, *RAID-5 Resizing.Drivers/md/raid5.c in the source code of Linux Kernel 2.6.32.9*, http://www.kernel.org/, Feb. 2010, accessed on Aug. 2012.

[11] 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.

[12] C. Kim, G. Kim, and B. Shin, "Volume Management in SAN Environment," *Proc. 8th Int'l Conf. Parallel and Distributed Systems (ICPADS)*, pp. 500-505, 2001.

[13] OLTP Application I/O and Search Engine I/O, *UMass Trace Repository*, http://traces.cs.umass.edu/index.php/Storage/Storage, Jun. 2007, accessed on Aug. 2012.

[14] Storage Performance Council, Defining, Administering, and Promoting Industry-Standard, Vendor-Neutral Benchmarks to Characterize the Performance of Storage Products. http://www.storageperformance.org/home, accessed on Aug. 2012.

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

[16] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan, "The HP AutoRAID Hierarchical Storage System," *ACM Trans. Computer Systems*, vol. 14, no. 1, pp. 108-136, Feb. 1996.

[17] G. Zhang, J. Shu, W. Xue, and W. Zheng, "SLAS: An Efficient Approach to Scaling Round-Robin Striped Volumes," *ACM Trans. Storage*, vol. 3, no. 1, pp. 1-39, Mar. 2007, article 3.

[18] C.B. Legg, "Method of Increasing the Storage Capacity of a Level Five RAID Disk Array by Adding, in a Single Step, a New Parity Block and N–1 New Data Blocks Which Respectively Reside in a New Columns, Where N Is At Least Two," Document Type and Number, U.S. Patent 6000010, Dec. 1999, accessed on Aug. 2012.
[19] S.R. Hetzler, "Data Storage Array Scaling Method and System with Minimal Data Movement," U.S. Patent 20080276057, Nov. 2008, accessed on Aug. 2012.
[20] W. Zheng and G. Zhang, "FastScale: Accelerate RAID Scaling by Minimizing Data Migration," *Proc. 9th USENIX Conf. File and Storage Technologies (FAST'11)*, pp. 149-161, Feb. 2011.
[21] C. Wu and X. He, "GSR: A Global Stripe-Based Redistribution Approach to Accelerate RAID-5 Scaling," *Proc. Int'l Conf. Parallel Processing (ICPP)*, pp. 460-469, 2012.
[22] C.R. Franklin and J.T. Wong, "Expansion of RAID Subsystems Using Spare Space with Immediate Access to New Space," U.S. Patent 10/033,997, 2006.
[23] S. Wu, H. Jiang, D. Feng, L. Tian, and B. Mao, "WorkOut: I/O Workload Outsourcing for Boosting the RAID Reconstruction Performance," *Proc. 7th USENIX Conf. File and Storage Technologies (FAST'09)*, pp. 239-252, Feb. 2009.
[24] A. Goel, C. Shahabi, S. Yao, and R. Zimmermann, "SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks," *Proc. 18th Int'l Conf. Data Engineering (ICDE'02)*, S. Chaudhuri, M. Carey, and H. Garcia-Molina, eds., IEEE CS Press, pp. 473-482, 2002.
[25] J. Alemany and J.S. Thathachar, "Random Striping News on Demand Servers," Technical Report TR-97-02-02, Univ. of Washington, 1997.
[26] J.R. Santos, R.R. Muntz, and B.A. Ribeiro-Neto, "Comparing Random Data Allocation and Data Striping in Multimedia Servers," *Proc. Measurement and Modeling of Computer Systems*, pp. 44-55, 2000.
[27] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, Distributed Data Placement Strategies for Storage Area Networks (Extended Abstract)," *Proc. ACM Symp. Parallel Algorithms and Architectures*, pp. 119-128, 2000.
[28] B. Seo and R. Zimmermann, "Efficient Disk Replacement and Data Migration Algorithms for Large Disk Subsystems," *ACM Trans. Storage (TOS) J.*, vol. 1, no. 3, pp. 316-345, Aug. 2005.
[29] 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)*, pp. 1-10, Apr. 2003.
[30] 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)*, pp. 1-10, Apr. 2004.
[31] S.A. Weil, S.A. Brandt, E.L. Miller, and C. Maltzahn, "CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data," *Proc. Int'l Conf. Super Computing (SC)*, pp. 1-12, Nov. 2006.
[32] 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. IEEE Int'l Conf. High Performance Computing (HiPC)*, pp. 1-10, Dec. 2011.
[33] J. Bonwick, *RAID-Z*, Nov. 2005, http://blogs.sun.com/bonwick/entry/raidz.
[34] Facebook, *HDFS RAID*, Nov. 2011, http://wiki.apache.org/hadoop/hdfs-raid, accessed on Aug. 2012.
[35] E. Pinheiro, W. Weber, and L. Barroso, "Failure Trends in a Large Disk Drive Population," *Proc. Conf. File and Storage Technologies (FAST'07)*, pp. 17-29, Feb. 2007.
[36] B. Schroeder and G. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" *Proc. Conf. File and Storage Technologies (FAST'07)*, pp. 1-16, Feb. 2007.

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

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

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