A New Parity-Based Migration Method to Expand RAID-5

Yu Mao, Jiguang Wan, Yifeng Zhu, *Member, IEEE*, and Changsheng Xie

Abstract—To expand the capacity of a RAID-5 array with additional disks, data have to be migrated between disks to leverage extra space and performance gain. Conventional methods for expanding RAID-5 are very slow because they have to migrate almost all existing data and recalculate all parity blocks. This paper proposes a new online expansion method for RAID-5, named parity-based migration (PBM). This method only migrates blocks that form a special parallelogram with one side consisting of only parity blocks. When adding m disks to a RAID-5 with n disks, PBM achieves the minimal data migration which only needs to move $m/(n+m)$ of all data blocks. Furthermore, no parity blocks are recalculated during the expansion. After expansion, although the RAID is not a standard RAID-5 distribution, the parity blocks are distributed evenly. Experimental results based on extensive trace-driven show that, on average, PBM can reduce the time of expansion by 73.6 percent while only reduces the performance of the expanded RAID by 1.83 percent when compared with Multiple-Device (MD), a toolkit provided in Linux kernel.

Index Terms—RAID-5, capacity expansion, data migration

1 Introduction

Redundant array of inexpensive disks Level 5 (RAID-5) is a popular disk array topology that provides a relatively low-cost solution to improve both performance and data reliability. With file system support for checksum, it can be resilient to single disk failure with multiple unrecoverable read errors during rebuilding. Although in some installations, data are mirrored between multiple storage servers (e.g., GoogleFS and Lustre) or multiple arrays in the same server (RAID 5 + 1), RAID-5 is still a standard implementation in many storage systems. Compared with RAID-6, though RAID-5 has low reliability, it has more capacity available and performance. When we consider more exotic forms of storage and memory (high-speed media, tiered, multi-server arrays, etc.), RAID-5 may provide enough data availability for a wide variety of use-cases.

As the volume of users' data grows at a phenomenal rate, it is often required to increase both the capacity and the performance of an existing RAID system to meet the increasing storage demands. When new disks are added to supply extra space and/or throughput, it is critical to avoid any downtime and minimize performance impact due to data movement in RAID reconstruction, and to eliminate the risk of data loss upon a disk failure during the expansion.

- *Y. Mao is with the Department of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. E-mail: routedhust@gmail.com.*
- *J. Wan and C. Xie are with Wuhan National Laboratory for Optoelectronics, Wuhan, China. E-mail: jgwan@mail.hust.edu.cn.*
- *Y. Zhu is with Electrical and Computer Engineering University of Maine.*

Manuscript received 22 Aug. 2013; revised 15 Oct. 2013; accepted 20 Oct. 2013. Date of publication 3 Nov. 2013; date of current version 16 July 2014. Recommended for acceptance by X.-H. Sun.
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.2013.279

The existing approaches [1], [2], [3] for RAID-5 expansion all rearrange data blocks to rebuild a standard RAID-5, in which data blocks and recalculated parity blocks are distributed across all of the disks in a round-robin manner. With a total of n disks in a RAID-5, block b resides on disk $b \bmod n$. However, when n changes, every data block has to be moved to a new location either on the same disk or on a different disk to reimplement the rigid modulus-based placement. As a result, reconstructing a new RAID-5 requires migrating nearly 100 percent of the data blocks. Moving such a large amount of data is always detrimental to the application performance.

This paper presents a new approach to expand a RAID-5. We have the following design goals.

- Online expansion with no downtime. Many systems have to ensure 7 × 24 availability of data services and thus these systems cannot be stopped to perform expansion.
- Balanced distribution of parity blocks. Since parity blocks tend to be hot spots (especially in write-intensive applications), uniformly distributing parity blocks across the array helps balancing the loads among all the disks after expansion.
- Minimal data movement. Theoretically, if m disks are added to a RAID-5 with n disks, the minimal fraction of data that has to be moved to achieve balanced data placement is $m/(n+m)$.
- No parity recalculation. If no new data is written into the storage systems, the expansion process should avoid recalculating the parity blocks since it can significantly prolong the expansion process.
- Incremental expansion. A RAID-5 may be expanded multiple times, which allows the RAID to continuously increase its storage capacity.

Our approach, named parity-based migration (PBM), uses two key techniques to achieve the above goals. The

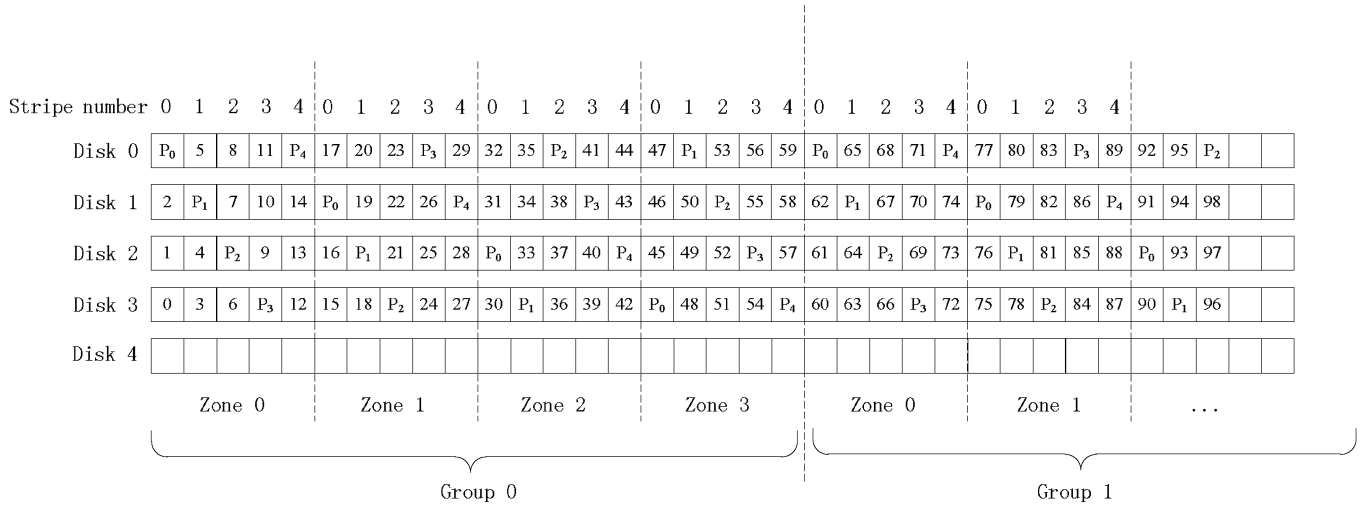


Fig. 1. The layout of *groups*, *zones*, and *stripes* when adding a new disk (disk 4 in this example) into an existing RAID-5 with four disks. In general, when adding m disks into a n-disk array, a set of $n+m$ consecutive *stripes* form a *zone*, and a set of n consecutive *zones* form a *group*.

first one is to selectively migrate data and parity blocks and reorganize them according to a specially predefined pattern. If all zones within a group (See Fig. 1) are virtually stacked on top of each other, PBM only migrates blocks that form a special parallelogram with one side consisting of only parity blocks (See Fig. 4). We prove that among all RAID-5 expansion schemes, PBM achieves the theoretical minimum in data migration. The second technique is to zero out blank blocks to avoid parity recalculation until a pre-existing data block is updated or a new data block is written onto the expanded RAID. The PBM-expanded RAID retains the property of tolerating one disk failure.

Throughout this paper, we use a RAID-5 with the layout of left-asymmetric parity distribution [4] as an example to illustrate the basic expansion process of PBM. One instance of such a RAID is shown in Fig. 1. The top 4 disks form a standard left-asymmetric RAID-5, where each stripe consists of a parity block and 3 data blocks. Parity blocks are distributed diagonally on these numbered disks, while data address decreases as the disk number grows in a stripe. (Notably, PBM is also applicable to a RAID-5 with other types of parity distributions.) As illustrated in Fig. 1, PBM divides the RAID into equal-sized groups, and the layout pattern of each group repeats across the array. When adding m new disks to an existing RAID-5 with an initial set of n disks, each group consists of n consecutive zones with $n+m$ stripes in each zone. In this paper we modify the definition of a stripe to be a column of blocks across all disks, and the stripe number starts with zero in each zone. Hence in a RAID-5 with n disks, each stripe includes $n-1$ data blocks and one parity block.

The rest of this paper is organized as follows. Section 2 presents the parity-based migration in detail. Section 3 describes experimental results and Section 4 discusses some related research. Section 5 concludes this paper and summarizes our future work.

2 PBM: Parity-Based Migration

The expansion of a RAID-5 in PBM involves two major steps: 1) migrating a set of associated blocks from old disks to new disks, and 2) allocating the blank blocks created in the first step to build new storage space. In the following, we first use two simple examples to illustrate our basic idea, then formally prove our results on the minimal data movement and perfectly balanced data redistribution. Multiple expansions are also discussed in this section. In supplementary file which is available in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.279, we discuss fundamental issues such as addressing, failure recovery, small writes and migration implementation.

In Table 1, we summarize the notations used in this paper.

2.1 Examples of RAID-5 Expansion

When new disks are added into a RAID-5 system, data have to be migrated from the initial set of disks (old disks) to the recently added ones (new disks). As mentioned in Section 1, many existing approaches [1], [2], [3] use modulo operation to determine on which disk a block b resides $(d = b \bmod n)$. When the total number of disks n

TABLE 1
Notations

Notation	Description
n	Number of disks in a RAID-5
m	Number of disks to be added
$E(n,m)$	The process of expanding a RAID-5 with n disks by adding m disks
g	Group number, starting at 0.
z	Zone number within a group, starting at 0.
s	Stripe number within a zone, starting at 0.
d	Disk number, $d = 0, 1, \ldots, n-1$
b	A block within a given group, addressed by (z,s,d)
$A(P)$	Associated blocks of parity block P (see Definition 2.1).
MT	Moving table of a group. Each table entry $(z,s,d) \rightarrow d^*$ denotes the migration of the block (z,s,d) to (z,s,d^*).

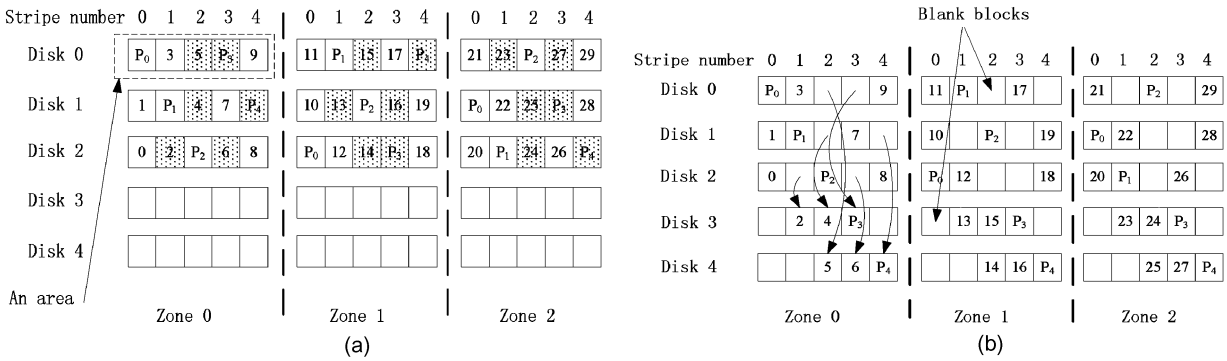


Fig. 2. Block layout of one group in $E(3,2)$. All blocks to be migrated are marked in (a). This is an example in which the blocks are balanced following redistribution. (a) Before migration. (b) After migration.

changes, nearly 100 percent of the data blocks have to be migrated. In contrast, in PBM we only need to migrate a minimal amount of data and parity blocks from old disks to new disks. In the following, we give two simple examples, $E(3,2)$ and $E(4,1)$, to illustrate the basic idea of parity-based migration, as shown in Figs. 2 and 3, respectively.

Example 1: Data Migration in $E(3,2)$. Assume that two new disks (disk 3 and 4) are to be added into a RAID-5 with three disks (disk 0, 1, and 2). Fig. 2a illustrates the block layout of Group 0 before expansion. For $E(3,2)$, each group consists of three zones, with five stripes in each zone. On each disk, five consecutive blocks in the same zone form an area. All areas in the old disks can be classified into two different categories: either with only one parity block or with two parity blocks. To uniformly distribute the parity blocks among all disks, one parity block in each area of the latter category will be migrated to a new disk in our design. Specifically, we migrate the one with a larger stripe number. For example, in zone 0, the parity block P_4 on disk 1 will be migrated.

When migrating a parity block, we also want to migrate its associated data blocks, which is defined below. Note that for simplicity, we only describe the migration process for the first group, which can be extended to other groups easily. Hence we identify a block by its zone number, stripe number, and disk number, i.e. (z,s,d), without mentioning its group number.

Definition 2.1. *For a parity block P identified by (z_p,s_p,d_p), if $s_p \in [n,n+m-1]$, then its associated block set is*

$$A(P)=\{(z_p,s_p-i,(d_p-2i) \bmod n)|0\le i\le n-1\}$$

where m is the number of new disks to be added.

Note that $P \in A(P)$ and $A(P)$ includes no other parity blocks except P. For example, in Fig. 2a, the marked parity block P_4 is (0, 4, 1), and its associated block set $A(P_4)=\{(0,4,1),(0,3,2),(0,2,0)\}$. We call all data blocks in $A(P_4)$ the associated data blocks of the parity block P_4.

$(d_p-2i) \bmod n$ is a strategy of data selection, which is not unique. In particular, any expression that selects exactly one block from each old disk with a distinct stripe number, including a unique parity block, satisfies our requirements. Here we choose such a strategy because of its simplicity.

In Fig. 2a, each zone has exactly two parity blocks that need to be migrated. Such parity blocks and their associated data blocks are marked in gray.

Note that when m is larger than 1, the target disk which associated blocks move to is arbitrary theoretically.

During the migration, we aim to balance the distribution of data and parity blocks among all the disks. This is achieved by migrating one associated block set onto an empty area on a new disk. Specifically, for a given parity block P identified by (z_p,s_p,d_p), we migrate $A(P)$ to the new disk whose disk number d is equal to s_p. For example, for the parity block P_4, we have $s_p=4$, thus the associated

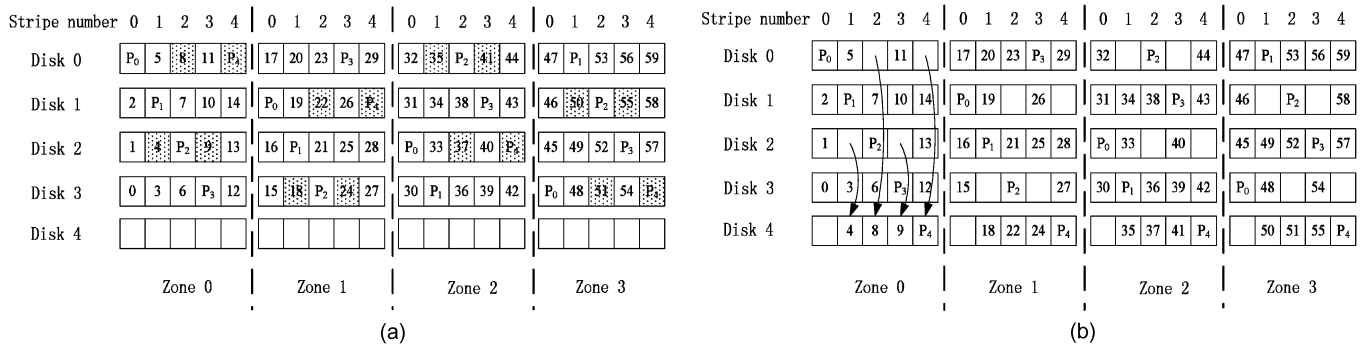


Fig. 3. Block layout of one group in $E(4,1)$. All blocks to be migrated are marked in (a). This is an example in which data redistribution are not balanced within a zone, but balanced across a group. (a) Before migration. (b) After migration.

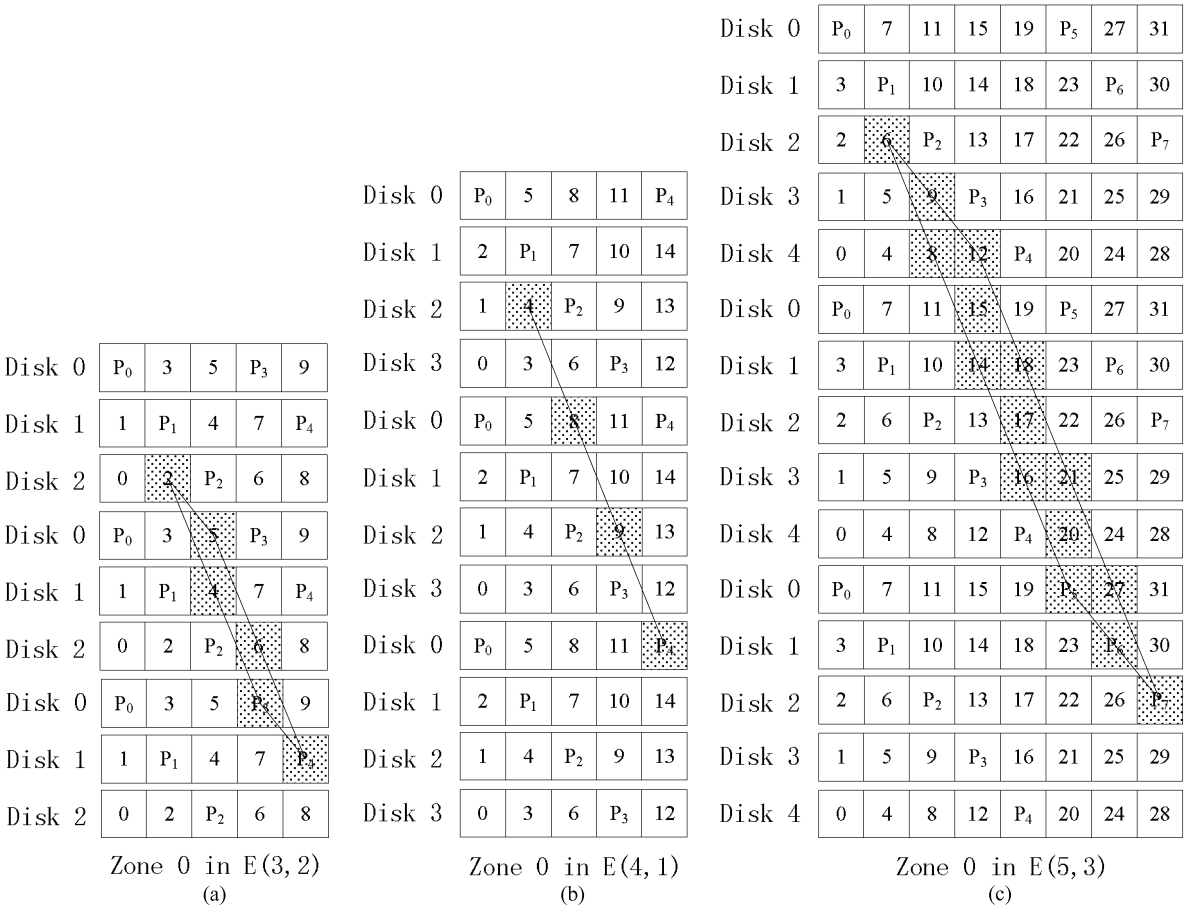


Fig. 4. All blocks to be migrated form a parallelogram with the bottom side consisting of only parity blocks when all zones in a group are stacked vertically. (a) The parallelogram in $E(3,2)$ (Example 1). (b) $E(4,1)$ is a special case when the parallelogram becomes a line (Example 2). (c) $E(5,3)$ is a more typical example in which all blocks migrated form a normal parallelogram.

block set {(0, 4, 1), (0, 3, 2), (0, 2, 0)}, i.e., {P_4, block 5, block 6}, are migrated to disk 4. During this process, each block keeps its original stripe number as illustrated in Fig. 2b. As a result, there is no need to recalculate the parity for each stripe after the migration.

Example 2: Data migration in $E(4,1)$. Fig. 3 shows the block layout before and after migration respectively, when adding a new disk to a RAID-5 with four disks. In this example, each group has four zones, and each zone has five stripes. In Fig. 3a, all blocks to be migrated in each zone are marked in gray. Although within a given zone, the blocks to be migrated are not uniformly distributed among the disks, within each group they indeed are. Specifically, in each group every disk has four such blocks. Hence after migration, all disks contain exactly the same amount of data and parity blocks, as showed in Fig. 3b.

These two examples respectively represent two different expansion scenarios, depending on the initial number of disks n being even or odd. If n is odd, blocks to be migrated are evenly distributed among disks within each zone. Otherwise if n is even, data redistribution is non-uniform within each zone, but uniform across each group.

Generally, each new stripe after migration keeps all existing data blocks of the corresponding stripe before migration. In addition, each disk after migration keeps the same number of data blocks, parity blocks and blank blocks, no matter n is odd or even. This characteristic can assure that every disk is accessed evenly after migration.

From the two examples above, we can see the amount of migrated blocks in a group is n^2m, and each group has $(n+m)\times n\times n$ original blocks (including data and parity blocks), so the migration ratio in Eqn. (1) shows our method moves the minimal data of whole RAID.

$$\begin{aligned}\text{Migration Ratio} &= \frac{\text{total num. of blocks migrated}}{\text{total num. of original blocks}} \\ &= \frac{n^2m}{(n+m)\times n\times n} \\ &= \frac{m}{n+m}. \end{aligned} \tag{1}$$

On the other hand, after migration every old disk in a group reserves n^2 blocks and every new disk also has n^2 blocks whichare non-empty. Therefore even distribution of blocks, including data blocks and parity blocks, is guaranteed.

2.2 Migration Algorithm

Once a block is identified to be migrated, we deploy a table-based approach to determine onto which disk this block

should be migrated. For a given parity block P, identified as (z_p, s_p, d_p), all of the associated blocks are migrated to the same new disk s_p. We build a small and simple table, called Moving Table (MT), to facilitate the calculation of the destination disk number. Each entry in the table is shown as follows.

$$\big(z_p, s_p - i, (d_p - 2i) \bmod n\big) \rightarrow s_p \tag{2}$$

for $\forall z_p \in [0, n-1]$, $\forall s_p \in [n, n+m-1]$ and $\forall i \in [0, n-1]$. Here the right hand side denotes the new disk number, which numerically equals the stripe number s_p, as mentioned in Example 1 in Section 2.1. Since d_p can be calculated as follows if the original RAID-5 is left-asymmetric distribution [4], [5]

$$d_p = (s_p + z_p m) \bmod n. \tag{3}$$

We have

$$\big(z_p, s_p - i, (s_p + z_p m - 2i) \bmod n\big) \rightarrow s_p \tag{4}$$

for $i = 0, 1, \ldots, n-1$.

Eq. (4) describes the mapping between a block and its new destination disk. Based on this mapping, we build the moving table, which is in fact very small and has very little memory footprint. Table 2 shows the MT for $z_p = 1$ in $E(3,2)$. Since all groups in the disk array are isomorphic, they all use the same moving table.

2.3 Allocating Blank Blocks

The second major step of expanding a RAID-5 is to allocate blank blocks to make more storage space available to users. In an expanded array, new data are allocated first to blank blocks with the lowest group number, the lowest zone number, the lowest stripe number and the lowest disk number in descending priority order. Fig. 5 gives a simple example in which three new data blocks are allocated to blank blocks.

In PBM, no blank block is used to recalculate the parity during writes. When a pre-existing data block, such as block 0, is updated, the parity block P_0 is recalculated and updated, i.e., $P_0 = (block0) \oplus (block1)$, as showed in Fig. 6a. When new data are written to block 10, as showed in Fig. 6b, P_0 is also recalculated. Specifically, we have $P_0 = (block0) \oplus (block1) \oplus (block10)$, or $P_0 = P_0^{old} \oplus (block10)$ for better performance. Note that the blank block on disk 4 is not used to recalculate P_0. All blank blocks are treated as zeros so that we do not have to read a blank block when calculating its corresponding parity.

2.4 Multiple Expansions

An expanded RAID-5 might have to be expanded again to further increase the capacity and/or the bandwidth. We

TABLE 2
Moving Table (MT) for $z_p = 1$ in $E(3,2)$

i \ s_p	3	4
0	$(1,3,2) \rightarrow 3$	$(1,4,0) \rightarrow 4$
1	$(1,2,0) \rightarrow 3$	$(1,3,1) \rightarrow 4$
2	$(1,1,1) \rightarrow 3$	$(1,2,2) \rightarrow 4$

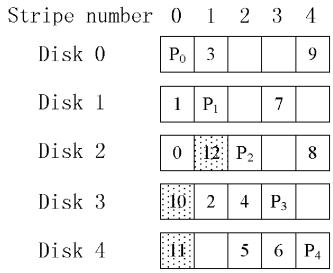


Fig. 5. New data block 10, 11, and 12 are allocated to three blank blocks that have the smallest stripe number and the smallest disk number in this zone.

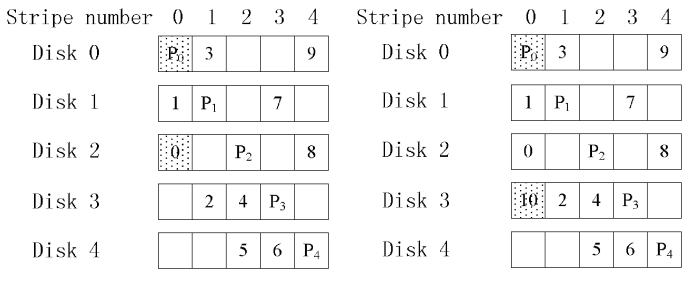
use the notation $E(n, m_1, \ldots, m_k)$ to represent the process of expanding a RAID-5 k times, with m_i new disks added in the ith expansion, for $i = 1, \ldots, k$. If the number of disks added in each expansion is arbitrarily selected, then the associated blocks of a given parity block reside on disks with irregular patterns, and thus in the expanded RAID data may not be evenly distributed among all disks. To avoid unbalanced distribution, we require that $m_1 = n$ and $m_{i+1} = n + \sum_{j=1}^{i} m_j$ for $1 \le i \le k-1$. In general, in the kth expansion, a total of $2^{k-1}n$ new disks are added.

With such a requirement, PBM can successfully maintain the uniform distribution even after a RAID has been expanded several times. Fig. 7 shows the block layout of $E(3,3)$ and $E(3,3,6)$. In Fig. 7b, blocks of the new storage space are marked. Even after being expanded twice, the expanded RAID $E(3,3,6)$ still has a data layout very similar to RAID-5 with left-asymmetric parity placement, as showed in Fig. 7c. We observe that all data and parity blocks of $E(3,3)$ are distributed uniformly among all disks in $E(3,3,6)$.

During multiple expansions, the moving table grows as the number of expansions increases. In the kth expansion, MT_k has $2^{2(k-1)}n^2$ entries. Considering the addressing, all the k MTs should be maintained in memory or disks. The total size of all MTs is:

$$\text{total size of MTs} = n^2 \sum_{i=1}^{k} 2^{2(k-1)} = \frac{n^2(2^{2k}-1)}{3}. \tag{5}$$

If k is a small number, the total size of MTs is not very large. For example, in $E(3,3,6)$, $k = 2$, $n = 3$, the total size of MTs is 45 entries.



(a) Updating an old data block (b) Writing a new data block

Fig. 6. Recalculate the parity block P_0 when (a) block 0 is modified, or (b) new block is written into block 10. No blank blocks is involved in parity recalculation. (a) Updating an old data block. (b) Writing a new data block.

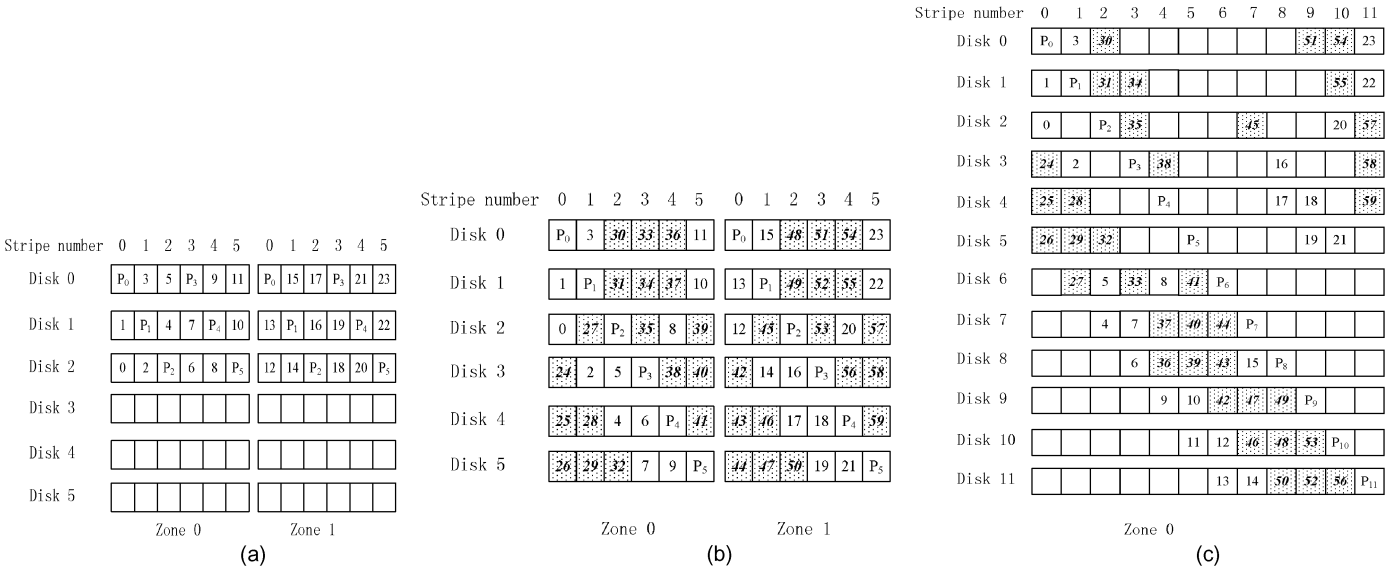


Fig. 7. Expanding a three-disk RAID two times. (a) Layout of $E(3,3)$ before expansion. (b) Layout of $E(3,3)$ after expansion. (c) Layout of $E(3,3,6)$: expanding the RAID again by adding six new disks.

3 Experiment Evaluation

We use trace-driven experiments on widely used Fedora 14 to compare PBM with three existing migration methods. The first one is Multi-Device (MD) Reshape toolkit [1], which has been included in standard Linux kernel. The other two are ALV [6] and GSR [7]. We develop a PBM expansion program that creates migration requests and metadata updates, performs address translations, and schedules migration requests and use *blktrace* and *btreplay* tools to issue I/O requests according to traces. Also, we develop the ALV and GSR programs based on MD to schedule outer I/O requests and migration requests. To reduce experimentation time, we use an acceleration factor of 10X.

The disks used in our experiments are based on Seagate_ST1000DM003. The default block size for most RAID-5 is 32-64 KB [8], [9]. In our experiments, we set the block size to be 64 KB. For 1000 GB will take a long time to migrate, we only use one volume to test our migration method, whose capacity is 10 GB. The key parameters of the disks are given in Table 3.

We use four different traces to evaluate our design. These traces include block level accesses, collected beneath the file system buffer cache but above the storage devices. Table 4 gives a summary of the trace characteristics.

- FIU traces are collected by FIU for the paper [10]. We use the second trace of four different end-user/developer home directories, which is named Home2. It is a 24-hour trace, which is downloaded from the Storage Networking Industry Association's trace repository [11].
- MSR-Cambridge is a serial of traces collected on production data servers by Microsoft researchers [12]. Only one trace is used in this study, which is Hm0 (hardware monitoring server, volume 0). It is a 7-day trace, which is also downloaded from the Storage Networking Industry Association's trace repository [13].
- Financial1 and Financial2 are two I/O block traces from OLTP applications running at two large financial institutions. These two traces are downloaded from the University of Massachusetts [14], and they are SPC (Storage Performance Council) format traces.

3.1 Migration Time

We compare the expansion time among PBM, MD, ALV and GSR when adding a different number of new disks into an existing RAID-5. During the online expansion process, the RAID also serves the I/O requests specified in the trace files. When the expansion completes, the experiment stops and no new I/O requests from trace files are issued. We also measure the expansion time when the migration is performed off-line, i.e., no traces are replayed during the expansion. Fig. 8 compares the migration time of four methods under four different workloads and under no workload (marked as *off-line*). When the number of new disks m increases from 1 to 3 or 4 in our experiments, more data are migrated onto new disks.

While more disks implies a larger degree of parallelism, it is interesting that the migration time of MD does not vary much as m changes, especially under workloads of Hm0 and Home2. As showed in Table 4, the I/O arrival rate in Hm0 is much lower than the other three workloads. As a result, the responder process in Hm0 takes a significantly

TABLE 3
Disk Parameters

Parameter	Value
Vender	Seagate
Disk Model	ST1000DM003
Disk Capacity	1000GB
Rotation Speed	7200 RPMS
RAID Block Size	64KB
Volume Capacity	10GB
Volume Start Address	2048B

TABLE 4
Trace Characteristics

Trace	R/W Ratio	Read Traffic (Sectors)	Write Traffic (Sectors)	Working Set (Sectors)	Total Requests	Requests per second
Home2	0.1	11,488,584	39,208,728	15,028,792	5,390,703	62.39
Hm0	0.55	20,890,647	42,943,920	5,235,621	3,993,316	6.6
Financial1	0.3	5,561,702	30,553,477	1,110,010	5,334,987	122
Financial2	4.67	13,882,741	3,810,800	909,660	3,699,194	90.2

lower fraction of disk bandwidth away from the migration process than it does in the other three workloads. Hence the migration process takes a much shorter time in Hm0. For Home2, although the I/O arrival rate is only about a half of Financial1, its write requests dominate the first 5000 s, so it takes more migration time, and longer response time than Financial1.

PBM and GSR only migrate $m/(n+m)$ of all data blocks, therefore the migration time is significantly shorter than ALV and MD.

The experimental results show that PBM can reduce the migration time by up to 91.83 percent, with an average time reduction of 73.6 percent across all experiments, compared to MD. On average, PBM can reduce migration time by 68.3 percent and 10.1 percent, compared to ALV and GSR respectively.

3.2 Performance in Migration

In this section we compare the performance in the migration process. While outer I/Os can affect the migration time, the migration operation also affects the I/O response time. We measure the response time in different traces with different n and m. Fig. 9 shows our expansion methods affect less outer I/O requests than MD. Generally, we can trade off the migration time for the I/O response time by adjusting the occurrences of migration. As described above, we decide to migrate a group when the I/O queue does not have enough requests. In our experiments, we set this threshold number to 64. If it is larger than 64, the migration time will be longer, but the response time will be shorter. Such influences are reversed if this number is smaller than 64.

The experimental results show that in the migration process, PBM and GSR have shorter outer I/O requests response time than the other two. On average, PBM can reduce response time by 19.5 percent and 8.1 percent, compared to MD and ALV respectively.

3.3 Performance after Migration

The block layout of a RAID expanded using PBM differs from standard RAID-5. In particular, we do not preserve a strict round-robin order in redistributing data and parity blocks. Specifically new blank blocks and pre-existing blocks are interleaved without any order in each stripe. Thus a large request that traverses the striping unit sequentially might visit the same disk twice sooner than a standard RAID-5. Hence in this section we evaluate the storage performance, in terms of the average response time, after the RAID is successfully expanded.

We compare the performance of RAID-5 expanded via PBM with that using MD or GSR. Note that after the expansion via MD, the new RAID is a standard RAID-5 with more disks (The same is true for ALV). Fig. 10 compares the performance after m disks are successfully added into an existing RAID-5 with n disks.

A large write may be split into multiple small writes(see Appendix C in supplementary file available online), which negatively impacts the performance. In some scenarios, however, our layout of blocks can positively influence the performance. As Fig. 11 shows, when a request needs to write blocks 3, 4, and 5, for our layout the parity block P_1 should be read, recalculated and written, thus in total 8 I/Os occur. In standard RAID-5 layout, however, P_0 and P_1 should both undergo a read-modify-write process, yielding 2 more I/Os than PBM, hence a longer response time. In addition, blocks 3 and P_1 cannot be read and written at the same time, thus parallelism is reduced. Fig. 10 shows that the performance degradation can be positive or negative. The average increase of response time is 1.83 percent.

Since GSR remaps the data blocks, two data blocks having consecutive address may reside at different physical addresses on two disks, or even at non-neighboring addresses on the same disk. This is the reason why GSR needs more response time than the other two methods in most cases. Our experiments show that on average, the

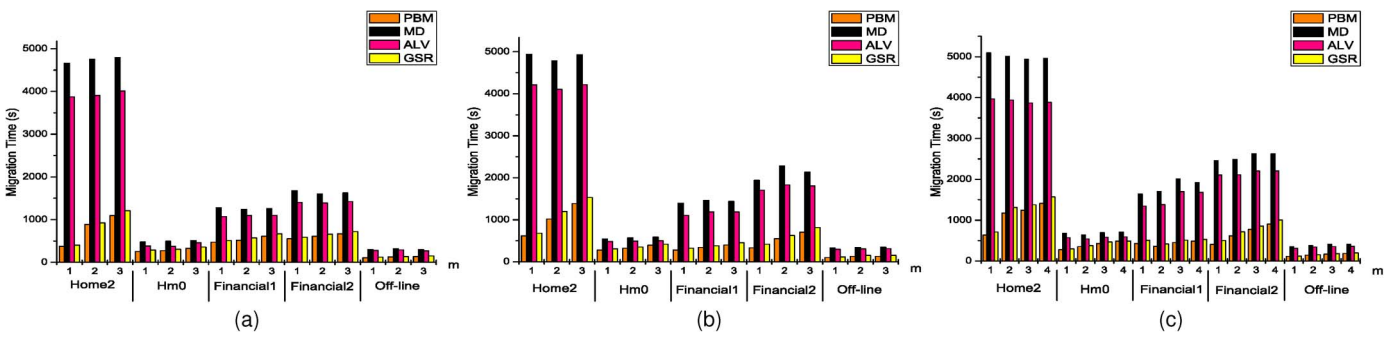


Fig. 8. Migration time comparison. (a) $n=3$, $m=1,2,3$. (b) $n=4$, $m=1,2,3$. (c) $n=5$, $m=1,2,3,4$.

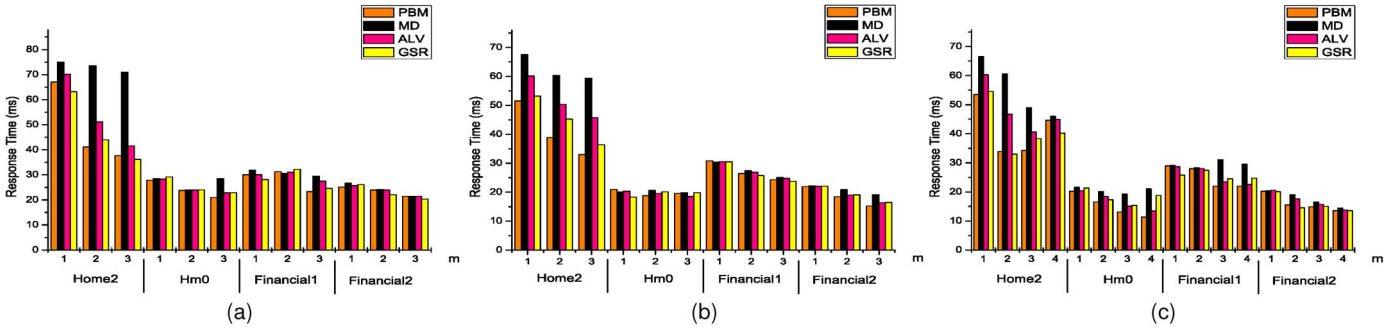


Fig. 9. Traces' response time comparison. (a) $n = 3$, $m = 1, 2, 3$. (b) $n = 4$, $m = 1, 2, 3$. (c) $n = 5$, $m = 1, 2, 3, 4$.

response time of GSR is 5.02 percent more than using standard RAID-5.

4 Related Work

Many studies have been conducted to expand an existing RAID, which can be classified into three categories: the ones that strictly preserve the round-robin order for all data and parity blocks, the ones that may violate the round-robin order occasionally, and the ones that place data and parity blocks randomly without any specific order.

4.1 Preserving a Round-Robin Order

Gradual assimilation [8] is a technique to add new disks to RAID-5 in an online manner to avoid downtime. It takes advantage of increasingly more storage idle times (due to added new disks) to reconstruct data on both old and new disks, stripe by stripe. The new disks become gradually available to serve user requests as the expansion proceeds.

SLAS [3] is a technique that improves the GA algorithm. It deploys reordering window and sliding window to control data migration.

ALV applies the basic idea of SLAS on RAID-5 [6]. Similarly to SLAS, ALV migrates the blocks in a sliding window, but it also recalculates the parity blocks in RAID-5. ALV has the same disadvantage as SLAS, that is, they both migrate almost all the data. According to [6], ALV can save redistribution time by at most 35.33 percent compared with MD-Reshape, while our method can save 31.39 percent of the migration time compared with MD-Reshape in the worst case.

FastScale [15] is recently proposed to solve the problem of migrating a large amount of data for RAID-0. It selects certain blocks to migrate and can achieve the minimal data migration. However, FastScale cannot deal with RAID-5 yet.

It has also been proposed to use the spare space of current RAID or additional disks to expand RAID in an online fashion [16]. This method uses spare space, including the spare space in the original disks or new spare disks, to accommodate incoming write requests. Data migration and new write requests operate separately. However, if the current RAID does not have enough spare space or no disks can accommodate incoming write requests, the expansion will fail, and more operations need to be done to guarantee data consistency.

Linux provides a tool named Multi-Device (MD) Reshape Tool [1] to support online capacity expansion. It performs well in round-robin stripped storage systems, including RAID-5. When adding m disks to the original RAID with n disks, MD reads $m + n - 1$ blocks, recalculates the parity, and then writes blocks and parity as a new stripe to the new RAID.

All of these methods which focus on RAID-5 move almost all the data blocks to reconstruct the RAID, so the expansion process is very slow.

4.2 Maintaining a Semi-Round-Robin Order

One patent proposed by Legg [17] aims to eliminate the need to rewrite the original data blocks and parity blocks on old disks during expansion. The key idea is to initialize new data blocks such that the exclusive-or of all blocks in a full stripe, including the parity block, is zero. The major disadvantage is that data and parity blocks are not evenly distributed among all disks. In addition, it has a large overhead since it requires a read and a write of every data block on the old disks.

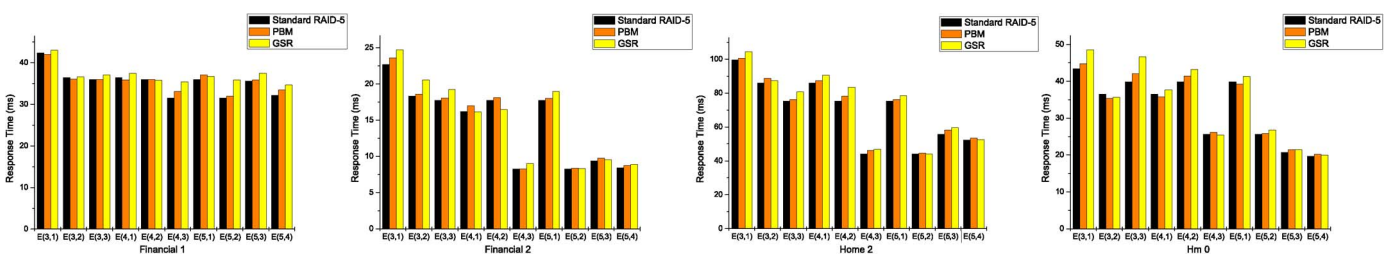


Fig. 10. Performance degradation after migration.

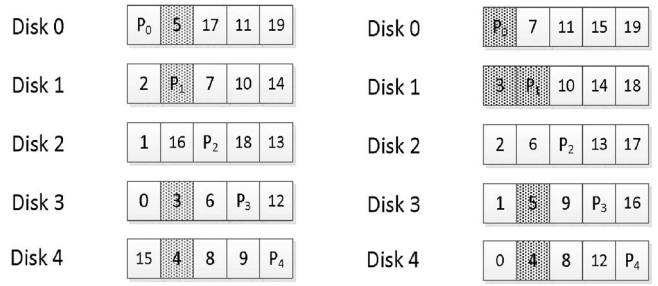


Fig. 11. Case of positive impact. Left graph is data layout after migration with PBM, and right graph is standard RAID-5 data layout.

Another patent developed by Corbett *et al.* [18] redistributes the parity blocks in a special pattern such that there is no need to recalculate parity or to move any data block when new disks are added. This scheme only works efficiently for arrays with a small number of disks, since a very large spare storage space is required for reconstruction when there are a large number of disks.

Yet another patent held by Hetzler [19] proposes to use unordered steps to dynamically select data blocks from old disks to rebuild a new stripe group on the expanded array. The new stripe pattern differs from standard RAID-5 since a parity block might be the exclusive-or of all data blocks in a vertical stripe or a diagonal stripe. However, some disks might hold more parity blocks than the other in the expanded array.

GSR [7] is a new RAID-5 scaling method. It divides all the stripes into three categories, which are Retained OUS, Remapped OUS and Destructed OUS, as Fig. 12 illustrated. GSR moves minimal data after scaling, but all the parity blocks need be recalculated. For example, in Fig. 12, all the Qs in the right graph are the new parity blocks. Recalculating parity causes more block-writes. For $E(n, m)$, GSR need to write $1/(n+m)$ more blocks than PBM, and its migration and user response time are worse than PBM. Moreover, GSR cannot perform multiple expansions.

4.3 Migrating Pseudo-Randomly

Random allocation methods, such as SCADDAR [20], [21], use a pseudo-randomized generator to generate a random number x for each block, and then place this block on disk $x \bmod N$, where N is the total number of disks. They do not need a directory system to keep track of the location of each block, because the same random number generator and the same seed are used for each block. Specifically, file names are used as the seed. When new disks are added, the random number generated for a block is mapped to a new number in a special way so that this number represents the disk on which this block resides after expansion. This approach requires file system level information that is typically not available at the RAID controller level. Thus although it is suitable for expanding the storage capacity of a server, it is challenging to use this method to expand a RAID.

5 Conclusion and Future Work

This paper proposes a new method to expand the capacity of an existing RAID-5 by adding new disks. This method,

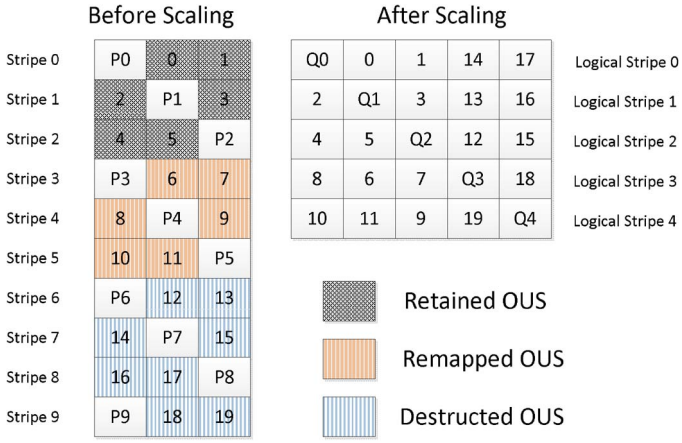


Fig. 12. Data layout of GSR.

called the parity-based migration (*PBM*), only migrates blocks that form a special parallelogram with one side consisting of only parity blocks. PBM is optimal in terms of the amount of migration data between new disks and old disks. Specifically, when adding m disks into an n-disk RAID-5, PBM only needs to migrate the minimal fraction of data, i.e., $m/(n+m)$, during the expansion process. Compared with the widely-used Linux toolkit *MD-Reshape*, PBM can save $n/(n+m)$ of data migration and thus can significantly reduce the negative impacts on the application performance during expansion. Furthermore, during the expansion, there is no need to recalculate the parity blocks unless a data block is updated. PBM also differs from standard RAID-5 in terms of data layout after migration. Data blocks are redistributed on all disks in a semi-round-robin order but new data and old data might be interleaved. However, experimental results show that the performance penalty incurred after migration is very small, less than 5 percent in most cases, with an average of 1.83 percent in response time.

There are three major limitations in PBM. First, for any n and m, the expanded RAID does not have the standard RAID-5 round-robin layout. The new layout combines new data and old data into one stripe, and new data can be stored anywhere on the disk, from low track to high track. Considering that the new data are likely to be accessed frequently, this kind of data layout can cause more seek time. Secondly, when a RAID-5 is expanded multiple times consecutively, to uniformly distribute data among all disks, PBM requires the number of new disks to be equal to the number of existing disks, i.e., $m = n$. Thirdly, the migration process of PBM is more negatively affected than MD if user requests are very busy. A dynamically adjusted migration granularity might be helpful to reduce such an impact. In the near future, we will address these three limitations.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable insights that have improved the quality of the paper greatly. This work supported by the National Basic Research Program (973) of China (No. 2011CB302303),

the National Natural Science Foundation of China (No. 60933002), and NSF under the grant No. 61300047 and the Fundamental Research Funds for the Central Universities' HUST2012QN101. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. J. Wan is the corresponding author.

REFERENCES

[1] N. Brown, *Online RAID-5 Resizing. Drivers/MD/Raid5.C in the Source Code of Linux Kernel 2.6.18*, Sept. 2006. [Online]. Available: http://www.kernel.org/

[2] J.L. Gonzalez and T. Cortes, ''Increasing the Capacity of RAID5 by Online Gradual Assimilation,'' in *Proc. Int'l Workshop I/Os—SNAPI*, 2004, pp. 17-24. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1162628.1162631

[3] G. Zhang, J. Shu, and W. Xue. (2007, Mar.). SLAS: An Efficient Approach to Scaling Round-Robin Striped Volumes. *ACM Trans. Storage* [Online]. *3(1)*, p. 3. Available: http://portal.acm.org/citation.cfm?id=1227838

[4] E. Lee and R. Katz. (1993, June). The Performance of Parity Placements in Disk Arrays. *IEEE Trans. Comput.* [Online]. *42(6)*, pp. 651-664. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=277289

[5] E.K. Lee and R.H. Katz, ''Performance Consequences of Parity Placement in Disk Arrays,'' in *Proc. 4th Int'l Conf. Architect. Support Programm. Lang. Oper. Syst.*, 1991, pp. 190-199. [Online]. Available: http://portal.acm.org/citation.cfm?id=106992

[6] G. Zhang, W. Zheng, and J. Shu. (2010, Mar.). ALV: A New Data Redistribution Approach to RAID-5 Scaling. *IEEE Trans. Comput.* [Online]. *59(3)*, pp. 345-357. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5276795

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

[8] J. Wilkes, R. Golding, C. Staelin, and T. Sullivan. (1996, Feb.). The HP AutoRAID Hierarchical Storage System. *ACM Trans. Comput. Syst.* [Online]. *14(1)*, pp. 96-108. Available: http://portal.acm.org/citation.cfm?id=225539

[9] J.L. Hennessy and D.A. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann, 2003.

[10] A. Verma, R. Koller, L. Useche, and R. Rangaswami, ''SRCMap: Energy Proportional Storage Using Dynamic Consolidation,'' in *Proc. 8th USENIX Conf. FAST*, 2010, pp. 267-280. [Online]. Available: http://www.usenix.org/events/fast10/tech/full_papers/verma.pdf

[11] FIU-Home2 Block I/O Trace, Storage Networking Industry Association, SNIA 2010. [Online]. Available: http://iotta.snia.org/traces/414

[12] D. Narayanan, A. Donnelly, and A. Rowstron. (2008, Nov.). Write Off-Loading: Practical Power Management for Enterprise Storage. *ACM Trans. Storage* [Online]. *4(3)*, pp. 1-23. Available: http://portal.acm.org/citation.cfm?doid=1416944.1416949

[13] MSR Cambridge Traces, Microsoft Research LTD, 2007. [Online]. Available: http://iotta.snia.org/tracetypes/3

[14] SPC-OLTP Application I/O, Storage Performance Council, University of Massachusetts, U Mass Trace Repository, 2007. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage

[15] W. Zheng and G. Zhang, ''Fastscale: Accelerate RAID Scaling By Minimizing Data Migration,'' in *Proc. 9th USENIX Conf. FAST*, 2011, p. 11. [Online]. Available: http://portal.acm.org/citation.cfm?id=1960475.1960486

[16] Expansion of RAID Subsystems Using Spare Space with Immediate Access to New Space, by C. Franklin and J.T. Wong. (2006, June 19). *U.S. Patent 20030 115 412*. [Online]. Available: http://www.freepatentsonline.com/y2003/0115412.html

[17] 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, by C.B. Legg. (1999, May 9). *U.S. Patent 6 000 010*. [Online]. Available: http://www.patentgenius.com/patent/6000010.html

[18] Semi-Static Distribution Technique, by P.F. Corbett, S.R. Kleiman, and R.M. English. (2007, Feb. 27). *U.S. Patent 07 185 144*. [Online]. Available: http://www.patentgenius.com/patent/6000010.html

[19] Data Storage Array Scaling Method and System with Minimal Data Movement, by S.R. Hetzler. (2008, Nov. 6). *20080 276 041*. [Online]. Available: http://www.faqs.org/patents/app/20080276041

[20] A. Goel, C. Shahabi, S. Yao, and R. Zimmermann, ''SCADDAR: An Efficient Randomized Technique to Reorganize Continuous Media Blocks,'' in *Proc. 18th Int'l Conf. Data Eng.*, 2002, no. 39, pp. 473-482. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=994760

[21] S.-Y. D. Yao, C. Shahabi, and P.-K. Larson. (2005, Apr.). Hash-Based Labeling Techniques for Storage Scaling. *VLDB J.* [Online]. *14(2)*, pp. 222-237. Available: http://www.springerlink.com/index/10.1007/s00778-004-0124-6

Yu Mao received the BS degree in computer science from Huazhong University of Science and Technology, China, in 2002, and the MS degree in University of South China, China, in 2007. He is currently pursuing the PhD degree in the Department of Computer Science, Huazhong University of Science and Technology, China. His research interests include storage, distributed system, parallel storage etc.

Jiguang Wan received the BS degree in computer science from Zhengzhou University, China, in 1996, and the MS and PhD degrees in computer science from Huazhong Univerisity of Science and Technology, China, in 2003 and 2007, respectively. He is currently an Associate Professor at Wuhan National Laboratory For Optoelectronics, Huazhong University of Science and Technology, China. His research interests include computer architecture, networked storage system, I/O and data storage architectures, and parallel and distributed system.

Yifeng Zhu received the BSc degree from the Huazhong University of Science and Technology, Wuhan, China, in 1998, and the MS and PhD degrees from the University of Nebraska, Lincoln, in 2002 and 2005, respectively. He is an associate professor at the University of Maine. His research interests include parallel I/O storage systems, and energy-aware memory systems. He served as the program committee of international conferences, including ICDCS and ICPP. He received the Best Paper Award at IEEE CLUSTER 07. He is a Member of the ACM, the IEEE, and the Francis Crowe Society.

Changsheng Xie received the BS and MS degrees in computer science from Huazhong University of Science and Technology, China, in 1982 and 1988, respectively. Presently, he is a Professor in the Department of Computer Engineering at Huazhong University of Science and Technology (HUST), China. He is also the Director of the Data Storage Systems Laboratory of HUST and the deputy director of the Wuhan National Laboratory for Optoelectronics. His research interests include computer architecture, disk I/O system, networked data storage system, and digital media technology. He is the vice chair of the expert committee of Storage Networking Industry Association (SNIA), China.