Hindawi Publishing Corporation
Journal of Electrical and Computer Engineering
Volume 2016, Article ID 7341735, 8 pages
http://dx.doi.org/10.1155/2016/7341735

Research Article

HS-RAID^2: Optimizing Small Write Performance in HS-RAID

Yongfeng Dong, Jingyu Liu, Jie Yan, Hongpu Liu, and Youxi Wu

School of Computer Science and Engineering, Hebei University of Technology, Xiping Road No. 5340, Beichen District, Tianjin 300401, China

Correspondence should be addressed to Jingyu Liu; liujy@hebut.edu.cn

Received 4 December 2015; Accepted 1 February 2016

Academic Editor: Hui Cheng

HS-RAID (Hybrid Semi-RAID), a power-aware RAID, saves energy by grouping disks in the array. All of the write operations in HS-RAID are small write which degrade the storage system's performance severely. In this paper, we propose a redundancy algorithm, data incremental parity algorithm (DIP), which employs HS-RAID to minimize the write penalty and improves the performance and reliability of the storage systems. The experimental results show that HS-RAID^2 (HS-RAID with DIP) is faster and has higher reliability than HS-RAID remarkably.

1. Introduction

RAID (Redundant Array of Independent Disks) [1, 2] combines multiple disk drives into a logical unit for the purposes of data redundancy or performance improvement. However, storage systems enlarge dramatically with the increasing of data. In April 2014, IDC reported that 4.4 ZB data was created by 2013, and the digital universe will be growing 40% a year into the next decade [3]. To meet the capacity demand, storage systems have grown to petabyte scale [4], and thousands of disks are deployed in storage systems. That caused a problem that cannot be ignored: high power consumption.

Many studies focused on power-saving of storage systems in recent years. The reason lies that higher power consumption leads to not only higher power costs in both storage systems and cooling systems, but also increasing operating temperature which can degrade the reliability and stability of the whole systems. Therefore, researchers proposed many kinds of strategies, such as DPM (Dynamic Power Management) algorithms, physical device level, and systems level [5].

DPM algorithm is proposed to turn disk devices into standby state to reduce power consumption during their idle period. However, DPM works only in the independent disk and does not work in RAID. At the physical device level, manufacturers are developing new energy efficient drives and hybrid drives. A hybrid drive combines NAND flash solid-state drive (SSD) with hard disk drive (HDD), with the intent of adding some of the speed of SSDs to the cost-effective storage capacity of traditional HDDs. The SSD in a hybrid drive acts as a cache for the data stored on the HDD by keeping copies of the most frequently used data on the SSD for improved overall performance and energy-saving.

At the system level, a number of integrated storage solutions such as MAID [6] and PERGAMUM [7] have emerged which are based on the general principle of transitioning the disks automatically to a low-power state (standby) after they experience some predetermined period of inactivity. PARAID [8] exploits the free space on the disks to duplicate data and uses a skewed striping pattern to adapt to the system load by varying the number of powered disks, thus needing no specialized hardware. eRAID [9] focuses on conventional disk-based mirrored disk array architectures like RAID 0. The power-saving effect is limited on parity redundant disk arrays like RAID 5. Hibernator [10] makes use of multispeed disk and abstracts the power saving problem into an optimization problem. It exploits the optimum solution in data migration between disks. EERAID [11] is another energy-efficient RAID system architecture which conserves energy by taking advantage of redundant information.

S-RAID [12] is an alternative RAID data layout for the kind of application that exhibits a sequential data access pattern. The data layout of S-RAID uses a grouping strategy that makes only part of the whole array active and puts the rest of the array into standby mode. Even in the sequential data

access storage system there are lots of random data accesses, and these degrade the performance of S-RAID dramatically. In our prework, we proposed an alternative RAID data layout based on S-RAID and HS-RAID [13], to avoid random data access effects and save the power consumption of the storage systems. HS-RAID is divided into two parts: RAID 1 and S-RAID 4/5. The first one is composed by SSDs for metadata storage, while the latter is composed by HDDs for data storage. HS-RAID is designed for the kind of applications that exhibit a sequential data access pattern, uses a grouping strategy that makes part of the whole array active, and puts the rest of the array into standby mode. Hence, HS-RAID can greatly reduce the power consumption and improve the reliability while still satisfying the I/O requirements of applications.

However, different RAID levels store data utilizing a variety of striping, mirroring, and parity techniques. RAID schemes based on parity improve the reliability of storage systems by managing a recovery parity disk. But parity calculations degrade performance especially performing small-write. The same problem also exists in HS-RAID whose parity schemes utilize RAID 4/5, and all writes in it are small writes.

This paper describes and evaluates a powerful parity algorithm, Data incremental parity algorithm (DIP), for eliminating the small write penalty in HS-RAID. DIP calculates parity data with the new data and the old parity and does not read the old data of the blocks which will be written in.

The remainder of this paper is organized as follows. Section 2 introduces the HS-RAID data layout. Following that, Section 3 gives detailed discussions of DIP algorithm and the write operation in HS-RAID^2. Then, Section 4 details data recovery in HS-RAID^2. The experimental results are presented in Section 5. Section 6 closes with a summary.

2. HS-RAID Data Layout

2.1. S-RAID. S-RAID [12] is an alternative RAID data layout for the kind of application that exhibits a sequential data access pattern. The data layout of S-RAID uses a grouping strategy that makes only part of the whole array active and puts the rest of the array into standby mode. Therefore, S-RAID can greatly reduce the power consumption and improve the reliability while still satisfying the I/O requirements of the application.

S-RAID [12] trades data transfer rate for energy efficiency and reliability and is suitable for the applications like video surveillance which requires moderate data transfer rate but large storage capacity and high reliability. These applications also exhibit a highly sequential data access pattern that S-RAID is optimized for.

However, even in the sequential data access application, there exist lots of random data accesses which degrade the performance of S-RAID dramatically. In order to avoid the adverse effects of random data access, HS-RAID was proposed in our prework.

2.2. HS-RAID. HS-RAID [12] includes two parts: RAID 1 and S-RAID. RAID 1 is composed of two SSDs, and S-RAID [10] is composed of a group of hard disks, as shown in Figure 1. Hard

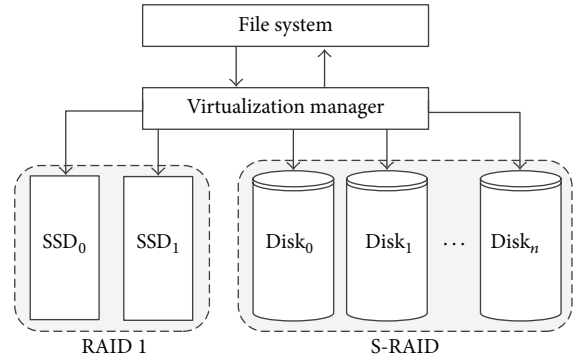

Figure 1: Configuration of Hybrid S-RAID.

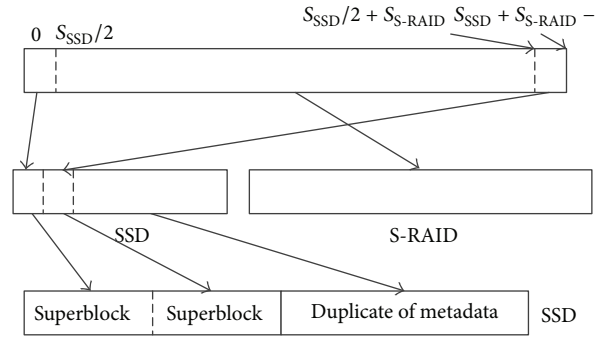

Figure 2: Address mapping.

disks are grouped in S-RAID and parallel in the group. The random I/O requests are mapped to RAID 1, while sequential I/O requests are mapped to S-RAID. When only one group of hard disks are busy under continuous data access mode, the others can be shut down or put into standby mode for energy-saving because of no requests on them. That can save energy of the whole storage systems and increase the costs slightly. HS-RAID is designed for applications whose I/O characteristics are sequential access.

RAID 1 which is composed of SSDs in HS-RAID is divided into three parts: two for storing superblock and one for metadata. The virtualization manager in Figure 1 maps the logical address to the physical address in the way shown in Figure 2. The addresses of two superblock partitions and S-RAID are identical logically, but the metadata partition is managed individually. The logical address $0 \sim S_{SSD}/2 - 1$ is mapped to the front of RAID 1, and it is 4 KB for superblock of the file system. $S_{SSD}/2 \sim S_{SSD}/2 + S_{S\text{-}RAID} - 1$ is mapped to S-RAID for data. $S_{SSD}/2 + S_{S\text{-}RAID} \sim S_{SSD} + S_{S\text{-}RAID} - 1$ is mapped to the second 4 KB of RAID 1 for superblock too. In fact, data of both superblocks is consistent.

In the metadata partition, the data is duplication of metadata which is in S-RAID. The metadata writes are done only if it is written in both the S-RAID and the metadata partition. The metadata, which is written once, never modified and read frequently in most sequential storage systems, is located in the metadata partition in order to avoid spinning up disks in standby mode.

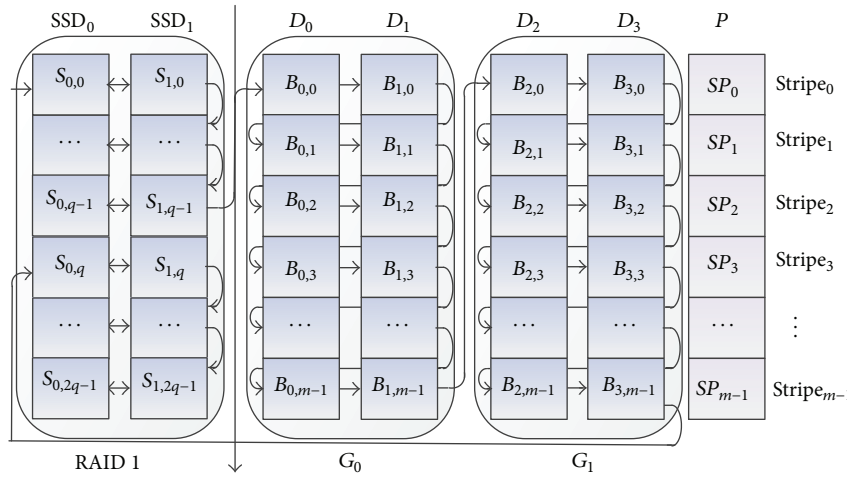

Figure 3: Data layout of HS-RAID 4.

2.2.1. HS-RAID 4. HS-RAID 4 is composed of a RAID 0 and a S-RAID 4 (shown as in Figure 3).

Figure 3 gives an example of data layout of HS-RAID 4. There are two SSDs (SSD_0 and SSD_1) and five HDDs (D_0, D_1, D_2, D_3, and P) in the storage system. One HDD (P) is the parity disk, and the other four HDDs (D_0, D_1, D_2, and D_3) are divided into two groups (G_0 and G_1) every two HDDs. The arrows in Figure 3 denote the data block sequence in the Logic Block Address (LBA) ascending order. $S_{i,j}$ denotes a data block in RAID 1 and $B_{i,j}$ denotes a data block in S-RAID, where i and j denote the SSD or HDD number and the stripe number, respectively.

Every LBA blkno in SSDs can be computed as

$$\text{blkno}\left(S_{i,j}\right) = \begin{cases} j, & 0 \le j < q-1; \\ T_s + j, & q \le j < 2q-1, \end{cases} \tag{1}$$

where T_s is the amount of blocks in the first part of RAID 1(q) and the total amount of blocks in S-RAID.

Obviously, blocks in both SSDs with the same offset have the same LBA:

$$\text{blkno}\left(S_{0,j}\right) = \text{blkno}\left(S_{1,j}\right). \tag{2}$$

Assume that there are r groups: $G_0, G_1, \ldots, G_{r-1}$ in S-RAID, and N_G is the amount of disks in each group. Then, we have $L_S = 0$, where L_S is the starting LBA of RAID 1, and $\text{blkno}(B_{0,0}) = q$. So the starting LBA of each group in S-RAID can be computed as

$$L_{G_k} = q + k * m * N_G \quad (k = 0, 1, \ldots, r-1). \tag{3}$$

Then, for every LBA blkno in the array, there exists a k that satisfies $L_{G_k} \le \text{blkno} < L_{G_{k+1}}$, and k can be calculated as

$$k = \left\lfloor \frac{\text{blkno} - q}{m * N_G} \right\rfloor. \tag{4}$$

Then, we can easily have mapping f:

$$\begin{aligned} f_{\text{stripe}_{\text{S-RAID}}}(\text{blkno}) &= \left\lfloor \frac{\text{blkno} - L_{G_k}}{N_G} \right\rfloor, \\ f_{\text{disk}_{\text{S-RAID}}}(\text{blkno}) &= \left(\text{blkno} - L_{G_k}\right) \bmod N_G + k \\ &\quad \times N_G, \\ f(\text{blkno}) &= B_{f_{\text{disk}_{\text{S-RAID}}}(\text{blkno}), f_{\text{stripe}_{\text{S-RAID}}}(\text{blkno})}. \end{aligned} \tag{5}$$

Every LBA blkno of $B_{i,j}$ can be computed as

$$\text{blkno}\left(B_{i,j}\right) = L_{G_k} + j \times N_G - k \times N_G. \tag{6}$$

2.2.2. HS-RAID 5. The fixed parity disk is the bottleneck of HS-RAID 4 because it not only degrades the performance but also reduces the system's reliability. Replacing S-RAID 4 with S-RAID 5 (shown as in Figure 4) in HS-RAID, we can get HS-RAID 5 which has uniformly distributed parity blocks among the disks.

HS-RAID 5 is composed of a RAID 0 and a S-RAID 5. For simplicity, when we discuss HS-RAID 5 in the rest of this paper, it means the S-RAID 5 partition. Figure 4 gives an example of a part of HS-RAID 5: S-RAID 5. Like in HS-RAID 4, we also use a grouping strategy that further divides the stripes into vertical groups in HS-RAID 5. There is no fixed parity disk in HS-RAID 5, instead of that we put parity blocks into different disks in each vertical stripe.

The arrows in Figure 4 denote the data block sequence in the Logic Block Address (LBA) ascending order. $B_{i,j}$ denotes a data block in S-RAID, where i and j denote the HDD number and the stripe number, respectively.

There are five HDDs (D_0, D_1, D_2, D_3, and D_4) in HS-RAID 5 which is shown in Figure 4. Five disks are divided into 2 groups. Each group may include different disks in different stripe because the parity blocks locate in different disk. For example, as shown in Figure 4, Group_0 includes D_0 and D_1

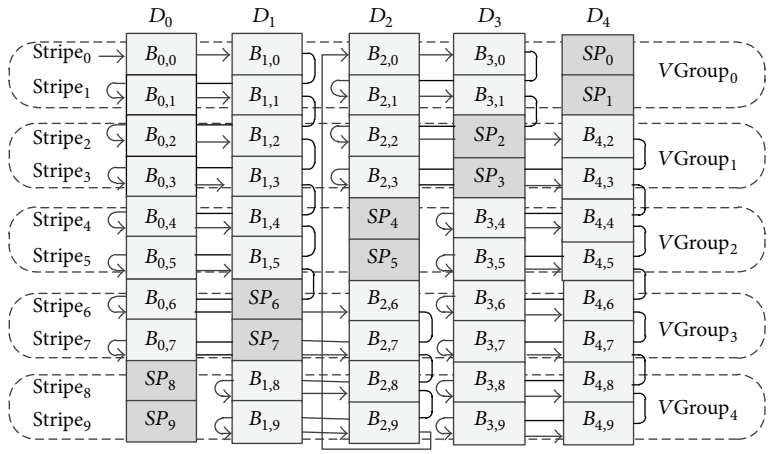

Figure 4: Data layout of S-RAID 5.

in Stripe$_1$, and Group$_1$ includes D_2 and D_3 in the same stripe. But in Stripe$_2$, the parity block is transferred into D_3. Group$_0$ includes the same disks as in Stripe$_1$, but Group$_1$ includes D_2 and D_4. For the same reason, in Stripe$_6$, Group$_0$ includes D_0 and D_2, and Group$_1$ includes D_3 and D_4.

3. HS-RAID^2: HS-RAID with New Redundancy Strategy

Before going into the detail of redundancy strategy, it is necessary to look at the write operation of HS-RAID.

Writing in the RAID 1 of HS-RAID is easy to understand, and it is not the focus of this paper, so it should be noted that when we mention the write operation in HS-RAID on the rest of the paper, it refers to the write operation in the S-RAID of HS-RAID.

3.1. Write Operation in HS-RAID. RAID 4/5 utilizes parity techniques to store data to enhance storage reliability. Write operations in RAID 4/5 are done only if the data and parities have been written. The parity is calculated from the data in the same address of the disks in RAID using XOR operation:

$$P = \bigoplus_i D_i, \tag{7}$$

where P and D_i are the parity disk and data disks, respectively.

When calculating the parity for RAID 4/5, the RAID controller selects a computation based on the write request size. The large-write parity calculation is

$$P_{\text{large-write}} = D_{\text{new}} \oplus D_{\text{remaining}}, \tag{8}$$

where $P_{\text{large-write}}$, D_{new}, and $D_{\text{remaining}}$ are the parity disk, the data on the disk(s) to be written, and the remaining disk(s), respectively. After computing the parity, the data and the new parity are written to the disks and the parity disk, respectively.

The small-write parity calculation is

$$P_{\text{small-write}} = D_{\text{new}} \oplus D_{\text{old}} \oplus P_{\text{old}}, \tag{9}$$

where $P_{\text{small-write}}$, D_{new}, and D_{old} are the parity disk, the data on the disk(s) to be written, the data to be replaced on the same disk(s), respectively. After computing the parity, the data and the new parity are written to the disks and the parity disk, respectively.

The main goal of HS-RAID is energy-saving by dividing disk into groups. Generally, only one group is active at the same time. In HS-RAID, all write operations are small write and select the computation "read-modify-write" to avoid spanning up other disks of groups that are put into standby mode.

Small-write parity in HS-RAID can be computed by XORing the old and new data with the old parity the same as in RAID. The parity calculation is

$$SP_{\text{new}} = D_{\text{old}} \oplus D_{\text{new}} \oplus SP_{\text{old}}. \tag{10}$$

When write requests are sent to HS-RAID, the corresponding disks are selected and made active if needed. Then, all selected blocks and parity blocks in the same stripes are read out. Lastly, HS-RAID recalculates the parity and writes the new data and the new parity back to disk. "Read-modify-write" brings write penalty severely and degrades the storage system's performance dramatically.

3.2. HS-RAID^2: HS-RAID with DIP. HS-RAID is designed for the kind of applications that exhibit a sequential data access pattern, such as video surveillance, continuous data protection (CDP), and virtual tape library (VTL). These systems have typical workload characteristics: write-once, read-maybe, and new writes unrelated to old writes.

We propose a new parity calculation algorithm applied to those applications workload characteristics: data incremental parity algorithm (DIP). In the rest of the paper, HS-RAID^2 is named for HS-RAID with DIP.

As Figure 5 shows, data in HS-RAID is always written in a new block in order. We set a pointerP_{LBA} as the last block that is written to. The initial value of P_{LBA} is "–1". In HS-RAID^2, the parity data is not calculated from all blocks in the stripe, but only blocks which are written to.

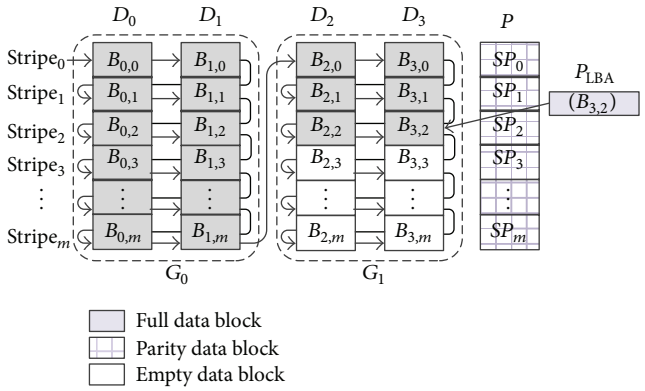

Figure 5: Sequential write in HS-RAID.

To maximize the benefit of DIP, write alignment is used: a buffer is set to collect data to be written large enough to write the whole stripe of a group.

Writing to the first group and the other groups in HS-RAID^2 is in different ways:

3.2.1. Write to the First Group. When writing to the disks in the first group, blocks and parity blocks are empty, so the parity calculation is

$$SP_{\text{new}} = \bigoplus_{i} D_i, \tag{11}$$

where SP_{new} and D_i are the parity and all the data that will be written to the array, respectively.

In blocks form, it is easy to transform formula (11) into formula (12):

$$SP_s = \bigoplus_{i=0}^{N_G-1} B_{i,s}, \tag{12}$$

where SP_s and $B_{i,s}$ are the parity block and the data block in disks of the first group, respectively. Disk number i and stripe number s can locate the block exactly. And N_G is the amount of disks in the group.

After calculating the parity, the data and parity in the array are written. We can safely say that it can minimize the write penalty since read operations are not needed.

Parity blocks need to be calculated while writing in the first group and the others, but they are different.

3.2.2. Write to the Other Groups. When writing to the disks in the other groups, parity blocks can be computed by XORing the old parity and the new data. Its calculation is

$$SP_{\text{new}} = D_{\text{new}} \oplus SP_{\text{old}}. \tag{13}$$

Suppose that write requests lie in group g. We can easily transform formula (12) into the following form:

$$SP_{s\_\text{new}} = \left(\bigoplus_{i=(g-1)*N_G}^{g*N_G-1} B_{i,s} \right) \oplus SP_{s\_\text{old}}. \tag{14}$$

Then, the data and parity to the array are written. Therefore, it can also minimize the write penalty, because old data does not need to be read while it does in RAID and HS-RAID.

4. Data Recovery in HS-RAID^2

Disk failure happens every day and even every hour in data centers. As with RAID 4/5, only one disk is allowed to fail in HS-RAID^2. When a disk in the array fails, it must be fixed or replaced in time to avoid another disk failing that causes all the storage system failure. HS-RAID^2 not only reduces the workload of write operations and minimizes the write penalty but also reduces the workload during the data recovery. In this section, we will discuss how HS-RAID^2 works during the data recovery after the failure disk is fixed or replaced and investigate the benefit in reducing the failure rate.

After the failure disk is fixed or replaced, how HS-RAID^2 recovers data depends on which disk fails. Generally, there are three cases.

(1) The Empty Disk. Although empty data disk fails rarely, it does. In this case, we need to replace the failure disk with a new one and initialize it without data written in. It notes that the data recovered from the array must be written into the new disk even if it means nothing in RAID and HS-RAID, because the parity of these arrays is calculated with all disks in it.

(2) The Disk with Data. If the failure disk is the one that has data written in, the data must be recovered immediately. Stripe numbers and disk numbers of blocks that need to be recovered can be calculated with the pointer P_{LBA}:

$$\begin{aligned} s &= \left\lfloor \frac{P_{\text{LBA}} - L_{G_g}}{N_G} \right\rfloor, \\ d &= \left(\left(P_{\text{LBA}} - L_{G_g} \right) \bmod N_{G_g} \right) + g * N_G, \end{aligned} \tag{15}$$

where N_G and g are the amount of disks in a group and the active group number, respectively. g can be calculated as

$$g = \left\lfloor \frac{P_{\text{LBA}}}{S * N_G} \right\rfloor, \tag{16}$$

where S is the amount of stripes in the array. L_{G_g} is the starting logical address of the group:

$$L_{G_g} = g * S * N_G. \tag{17}$$

Then, how to recover the data depends on whether the disk is in group g or not.

(a) The Failure Disk Is in Group g. Data blocks from stripe_0 to stripe_s should be recovered if the failure disk is in group g. Data is recovered by XORing the blocks from group_0 to

$group_g$ and the parity in the same stripe. Suppose that the number of the failure disks is I; then,

$$B_{i,m} = B_{0,m} \oplus B_{1,m} \oplus \cdots \oplus B_{i-1,m} \oplus B_{i+1,m} \oplus \cdots \oplus B_{(g+1)*N_G-1,m} \quad (0 \le m \le s), \tag{18}$$

where m is the stripe number. As shown in Figure 5, suppose that disk D_3 fails while data is written to $B_{2,2}$ and $B_{3,2}$. After the failure disk is replaced, $B_{3,0}$, $B_{3,1}$, and $B_{3,2}$ will be recovered:

$$B_{3,m} = B_{0,m} \oplus B_{1,m} \oplus B_{2,m} \oplus SP_m \quad (m = 0, 1, 2). \tag{19}$$

The rest stripes in the failure disk are empty, so the blocks in these stripes need not be recovered.

(b) The Failure Disk Is Not in Group g. If the failure disk does not belong to group g, all the blocks in the disk need to be recovered. Suppose that the max. stripe number is S. Data from $stripe_0$ to $stripe_s$ are recovered by XORing the blocks from $group_0$ to $group_g$ and the parity block in the same stripe using the formula (18). Data from $stripe_{s+1}$ to $stripe_S$ are recovered by XORing the blocks from $group_0$ to $group_{g-1}$ and the parity block in the same stripe:

$$B_{i,m} = B_{0,m} \oplus B_{0,m} \oplus \cdots \oplus B_{i-1,m} \oplus B_{i+1,m} \oplus \cdots \oplus B_{(g+1)*N_G-1,m} \oplus SP_m \quad (s < m \le S). \tag{20}$$

Suppose that disk D_1 fails; data are recovered as follows:

$$B_{1,m} = \begin{cases} B_{0,m} \oplus B_{2,m} \oplus B_{3,m} \oplus SP_m & (m = 0, 1, 2) \\ B_{0,m} \oplus SP_m & (m = 3, 4). \end{cases} \tag{21}$$

(3) Parity Disk. If the parity disk fails, the disk should be replaced and the parity data should be recalculated. The parity data from $stripe_0$ to $stripe_s$ is recalculated with the data of $group_0$ to $group_g$, and the parity data from $stripe_{s+1}$ to $stripe_S$ is recalculated with the data of $group_0$ to $group_{g-1}$ in the same stripe:

$$SP_m = B_{0,m} \oplus B_{1,m} \oplus B_{(g+1)*N_G-1,m} \quad (0 \le m \le s),$$
$$SP_m = B_{0,m} \oplus B_{1,m} \oplus B_{g*N_G-1,m} \quad (s < m \le S). \tag{22}$$

For example, if parity disk P (as shown in Figure 4) fails, the parity data will be recalculated as follows:

$$SP_m = \begin{cases} B_{0,m} \oplus B_{1,m} \oplus B_{2,m} \oplus B_{3,m} & (m = 0, 1, 2) \\ B_{0,m} \oplus B_{1,m} & (m = 3, 4). \end{cases} \tag{23}$$

Above all, when a disk in HS-RAID^2 fails, data recovery need to read less data and do less calculation than in HS-RAID or RAID. It is easy to understand that it can reduce the recovery time, degrade the energy-consuming, and improve the reliability of the storage system.

Table 1: The characteristic of the server.

Description	Value
Model	HP ProLiant DL180
CPU	Xeon E5620, 2.4 GHz
Interface	SATA/SAS

Table 2: The characteristics of the SSD.

Description	Value
SSD model	Crucial CT128M4SSD2
Standard interface	SATA3.0
Size	128 G
Average access time	<0.1 ms
Startup power	0.15 watts

Table 3: The characteristics of the HDD.

Description	Value
Model	Seagate ST3500630AS
Interface	SATA 2.0
Rotational speed	7200 rpm
Size	500 G
Active power	13 W
Idle power	9.3 W
Standby power	0.8 W
Spin-up time	15 s

5. Performance Comparison

The design target of HS-RAID^2 is to reduce the write penalty in storage systems. As described before, this target is achieved by a new parity algorithm, DIP algorithm.

The prototype of HS-RAID^2 consists of two SSDs and twelve HDDs, all the devices' parameters are shown in Tables 1, 2, and 3. It runs the operating system Linux kernel 2.6.35.32. The block size is 64 KB. To compare the performance, the current HS-RAID prototype is the same as HS-RAID^2.

Different applications have different performance requirements. To be fit for the applications' requirements, HS-RAID^2 and HS-RAID could set varying amount of disks per group. Each of them has three grouping scales: 1 disk per group, 2 disks per group, and 3 disks per group. The performances of both HS-RAID^2 and HS-RAID are evaluated in different grouping scale and different write sizes. The results are shown in Figures 6, 7, and 8.

Figure 6 shows the transfer rate speedup of HS-RAID^2 with 1 disk per group compared to HS-RAID with the same disk per group. These are the 100% sequential write workloads in the experiments in the 16 KB to 4096 KB range. When the write size is no larger than 64 KB, both HS-RAID^2 and HS-RAID have poor performances, and the transfer rate of HS-RAID is less than 5 MB/s especially. Even in HS-RAID^2, the transfer rate is no more than 20 MB/s. However, it gets the maximum speedup, which is 578%, when the write size is 16 KB. It illuminates that DIP takes the most advantage at that time because the write alignment and read-ahead do

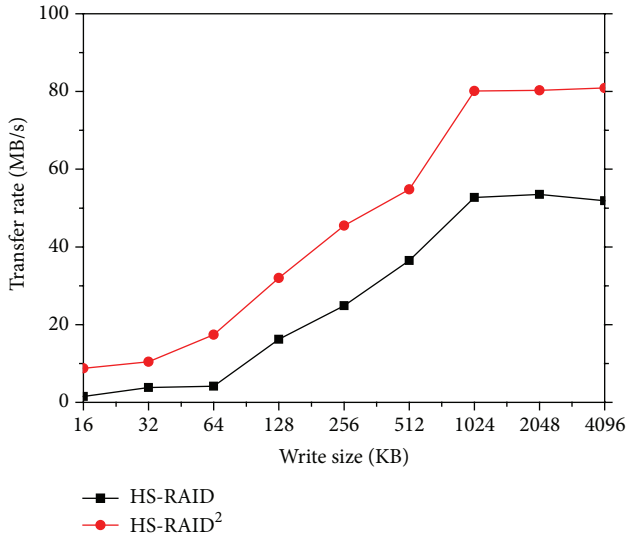

Figure 6: Performance of HS-RAID and HS-RAID^2 (1 disk per group).

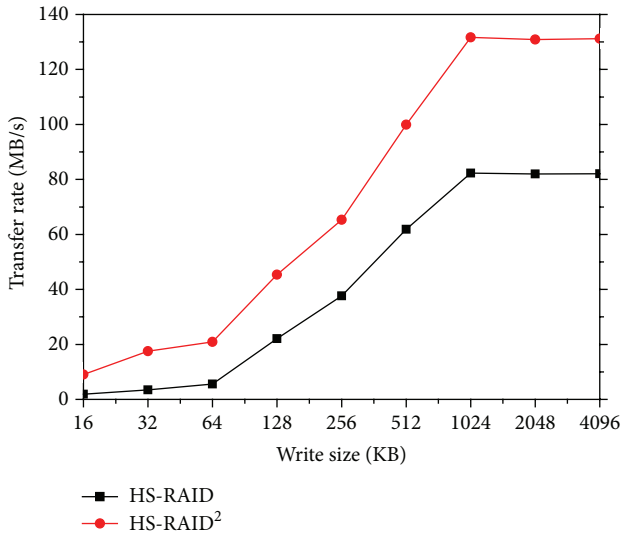

Figure 7: Performance of HS-RAID and HS-RAID^2 (2 disks per group).

their best work. When write size is larger than 64 KB, the transfer rate of both HS-RAID^2 and HS-RAID is improved dramatically. The peak value is 82.5 MB/s when write size is 1 MB in HS-RAID^2 while it is 52.6 MB/s in HS-RAID. When write size is larger than 1 MB, the speedup ratio is less than 60%, but the speedup is approximately 30 MB/s in value.

Figure 7 shows the transfer rate speedup of HS-RAID^2 with 2 disks per group compared to HS-RAID with the same disks per group. These are also the 100% sequential write workloads in the experiments in the 16 KB to 4096 KB range. It gets the maximum speedup, which is 506%, when the write size is 32 KB. When write size is larger than 64 KB, the transfer rate of both HS-RAID^2 and HS-RAID is improved

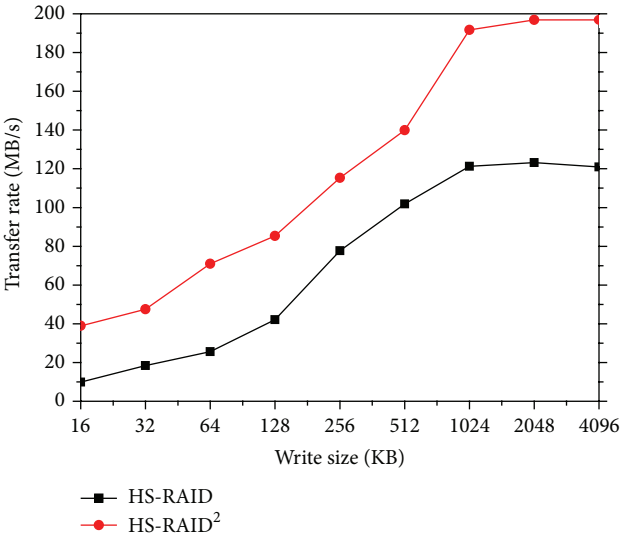

Figure 8: Performance of HS-RAID and HS-RAID^2 (3 disks per group).

dramatically. The peak value is 132.5 MB/s when write size is 1 MB in HS-RAID^2, while it is 80.2 MB/s in HS-RAID.

Figure 8 shows the transfer rate speedup of HS-RAID^2 with 3 disks per group compared to HS-RAID with the same disks per group. It gets the maximum speedup, which is 397%, when the write size is 32 KB. When write size is larger than 128 KB, the transfer rate of both HS-RAID^2 and HS-RAID is improved dramatically. The peak value is 197.1 MB/s when write size is 2 MB in HS-RAID^2 while it is 123.2 MB/s in HS-RAID.

The smaller the write request size is, the slower the systems transfer rate is, and the larger the write request size is, the more improved the system performance is. When writing to HS-RAID^2 and HS-RAID with 3 disks per group in the size 16 KB, the transfer rate of HS-RAID^2 is 565% faster than HS-RAID. It is the max speedup factor. When writing to HS-RAID^2 and HS-RAID with 2 disks per group in the size 1 MB, the transfer rate of HS-RAID^2 is 52.3% faster than HS-RAID. It is the min. speedup factor.

Above all, it is obvious that the performance of HS-RAID^2 has been greatly improved compared to HS-RAID.

6. Conclusion

HS-RAID saves the power consumption of storage systems by dividing disks in the array into groups. All of the write operations in HS-RAID are small write in order to avoid spanning up disks in standby mode. Small write degrades the storage system's performance.

HS-RAID^2 has the same architecture as HS-RAID and employs different parity calculation algorithm: DIP. DIP minimizes the write penalty in HS-RAID and improves the performance and reliability of storage systems. The experimental results show that HS-RAID^2 is faster and has higher reliability than the traditional method.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the Youth Foundation of Education Commission of Hebei Province of China (QN2014192), the National Natural Science Foundation of China (61571180), the Research Foundation of Education Commission of Hebei Province of China (ZD2014009), and the Science and Technology Project of Hebei Province of China (15210325).

References

[1] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: high-performance, reliable secondary storage," *ACM Computing Surveys*, vol. 26, no. 2, pp. 145–185, 1994.

[2] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '88)*, pp. 109–116, Chicago, Ill, USA, 1988.

[3] 2014, http://www.emc.com/leadership/digital-universe/index.htm.

[4] A. Thusoo, Z. Shao, S. Anthony et al., "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*, pp. 1013–1020, Indianapolis, IN, USA, June 2010.

[5] E. Otoo, D. Rotem, and S.-C. Tsao, "Dynamic data reorganization for energy savings in disk storage systems," in *Scientific and Statistical Database Management: 22nd International Conference, SSDBM 2010, Heidelberg, Germany, June 30–July 2, 2010. Proceedings*, vol. 6187 of *Lecture Notes in Computer Science*, pp. 322–341, Springer, Berlin, Germany, 2010.

[6] D. Colarelli and D. Grunwald, "Massive arrays of idle disks for storage archives," in *Proceedings of the ACM/IEEE Conference on Supercomputing*, pp. 1–11, Baltimore, Md, USA, 2002.

[7] M. W. Storer, K. M. Greenan, E. L. Miller et al., "Pergamum: replacing tape with energy efficient, reliable, disk-based archival storage," in *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, pp. 1–16, Berkeley, Calif, USA, 2008.

[8] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning, "PARAID: a gear-shifting power-aware RAID," *ACM Transactions on Storage*, vol. 3, article 13, 2007.

[9] J. Wang, H. Zhu, and D. Li, "eRAID: conserving energy in conventional disk-based RAID system," *IEEE Transactions on Computers*, vol. 57, no. 3, pp. 359–374, 2008.

[10] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: helping disk arrays sleep through the winter," *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 177–190, 2005.

[11] D. Li and J. Wang, "EERAID: energy efficient redundant and inexpensive disk array," in *Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, pp. 1–14, ACM, September 2004.

[12] X. Li, Y.-A. Tan, and Z.-Z. Sun, "Semi-RAID: a reliable energy-aware RAID data layout for sequential data access," in *Proceedings of the IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST '11)*, pp. 1–11, Denver, Colo, USA, May 2011.

[13] L. Jingyu, Z. Jun, L. Yuanzhang, S. Zhizhuo, W. Wenming, and T. Yu-An, "Hybrid S-RAID: an energy-efficient data layout for sequential data storage," *Journal of Computer Researchs and Development*, vol. 50, no. 1, pp. 37–48, 2013 (Chinese).