Partial Parity Cache and Data Cache Management Method to Improve the Performance of an SSD-Based RAID

Ching-Che Chung, *Member, IEEE*, and Hao-Hsiang Hsu

***Abstract*— In this paper, a partial parity cache and data cache management method is presented for reducing the parity updating cost of a solid-state disk (SSD) based redundant array of inexpensive disk (RAID) system, thereby which the input/output (I/O) performance of the RAID system can be improved. SSDs have many advantages compared to hard disk drives. However, it is not advisable to directly add SSDs into a RAID system because doing so will decrease the performance and the life-time of the SSDs. In the RAID-5 system, parity generation includes read and write operations to the SSDs. Whenever there is a new write request to the RAID, the related parity must be updated and written to the SSDs. Such frequent parity updates result in poor RAID performance and shortens the life-time of the SSDs. This paper combines the prior methods and the proposed efficient buffer management method with a data cache. The proposed method reduces the number of read and write operations for generating parities in the RAID system. Experimental results show that the I/O performance of the RAID-5 system can be improved by 76% by using the proposed method.**

***Index Terms*— Flash memory, hard disk drive (HDD), redundant array of independent disks (RAIDs), solid-state disk (SSD).**

I. Introduction

MANY advancements have been made in the design of central processing units (CPUs) for personal computers (PCs). The number of cores in a CPU has increased from one to many, and thus there has been a clear improvement in the performance of CPUs. Further, the other components of a PC, such as dynamic random access memories (DRAMs) and video cards, have also undergone substantial improvement. However, because of certain physical limitations, the improvement of hard disk drivers (HDDs) (which are composed of platters and require a head actuator mechanism) is not evident. In recent years, solid-state disks (SSDs) have become increasingly popular because of the relatively low price of flash chips [1].

As shown in Fig. 1, SSDs consist of many flash memory chips and do not have a head actuator mechanism. All components of the SSDs are electronic, so SSDs have many advantages over HDDs, e.g., better shake resistance, lower power consumption, and faster performance.

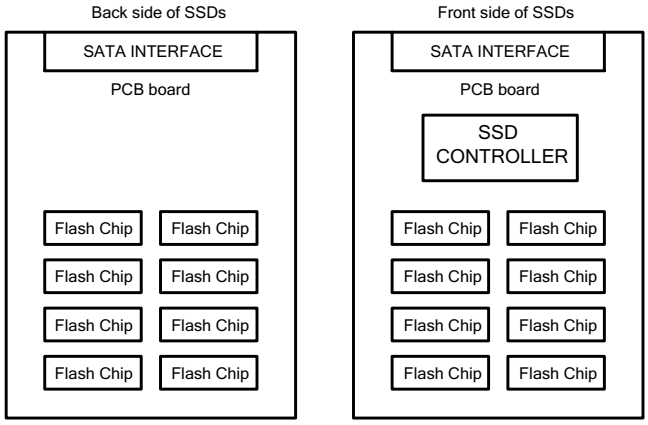

Fig. 1. Front side and back side of SSDs.

TABLE I
SPECIFICATIONS OF THE NAND FLASH CHIP [9]

Hynix 32GB NAND Flash Chip	
Data Integrity	100,000 erase cycles
Page Read	0.025 ms
Page Program Time	0.2 ms
Block Erase Time	2 ms

Flash memory is the basic component of the SSD. Flash memory has the following characteristics: First, the unit for read and write operations is a page, but the unit for an erase operation is a block [2]–[4]. Therefore, the speeds in the different operations are greatly different, as shown in Table I. Second, the same physical page can be written upon only once after each erase operation. Third, each block has a limited number of erase times [11], [12].

When data are written to the flash memory, the amount of free pages becomes small, and thus the flash controller must erase a block to recycle free pages. Before the controller erases a block, the controller must copy the valid data in the block to another free block; this operation is called garbage collection (GC). Table I shows that the block erase time is 10 times longer than the page program time. Therefore, the write operation, erase block operation, and GC take a considerable number of cycles to complete. As a result, the method for decreasing the number of write operations is very important for the SSDs. When the number of write operations

Manuscript received February 6, 2013; revised June 9, 2013; accepted July 25, 2013. Date of publication August 15, 2013; date of current version June 23, 2014. This work was supported by the National Science Council of Taiwan under Grant NSC-100-2221-E-194-051.

The authors are with the Department of Computer Science and Information Engineering, National Chung Cheng University, Chia-Yi 621, Taiwan (e-mail: wildwolf@cs.ccu.edu.tw; jack@s3lab.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2275737

is decreased, the life-time of an SSD is extended and the input-output (I/O) processing speed is improved.

In the data center, HDDs are used in a redundant array of inexpensive disks (RAID) as the storage system because of the low price of HDDs. However, the power consumption and the heat dissipation of HDDs are critical problems. For instance, 40% of the entire power consumption is required for cooling down the data center [5]. Because SSDs are more expensive than HDDs with the same capacity, SSDs cannot replace HDDs even though they consume less power and have faster I/O performance than HDDs. Therefore, there are a few companies that are developing a RAID controller for SSDs. In the future, when the price of flash chips decreases substantially, SSDs can be used in the data center as the storage system in order to reduce the costs of cooling down the data center and to increase the I/O processing speed. Further, the data center always uses the RAID technique to enhance performance and ensure data integrity.

Some companies such as Hitachi and Samsung have proposed solutions that involve the use of SSDs in the data center [6], [7]. Hitachi [6] has developed a data center by using a serial-attached SCSI (SAS) interface and a fiber channel (FC) with SSDs. Reinsel and Janukowicz [7] have mentioned have mentioned that, in the future, the challenges of the data center are the reliability and price of SSDs. SSDs can save energy and enhance the speed performance, as analyzed in [8].

There are products with RAID-0 SSDs in the market. The I/O processing speed of RAID-0 is very high, but RAID-0 is not adequately reliable. When one of the flash chips is damaged, the stored data are lost. From the perspective of reliability, RAID-4 and RAID-5 are better solutions, as they can use parity to recover data when one of the storage devices gets damaged.

Table I shows the specifications of a NAND flash memory chip [9]. The program time and the erase time are very slow as compared to the read time. However, in RAID-4 or RAID-5, there are many write operations for parity updates. This issue adversely affects the I/O processing speed of the entire RAID system.

As long as new data are written to the storage system of RAID-4 and RAID-5, the parity must be updated with read operations to generate a new parity. Then, the new parity will be written to the storage system. The storage system must reduce the parity generation overheads and parity write times. Hence, we need an efficient parity cache management scheme to increase the I/O processing speed of the RAID system while ensuring that the system is reliable.

This paper combines the prior partial parity cache (PPC) method [10] and the proposed efficient buffer management method with a new data cache to merge parities. The experimental results show that both read and write operations are decreased by using the proposed method. Further, we add a special data buffer that retains the old data. This data buffer can reduce the read operations from SSDs for a partial parity update, so that the I/O processing speed can be further improved. An efficient buffer management method can reduce the parity write times, and the proposed data cache can reduce the number of read operations required for parity updates. In addition, the proposed method also extends the life-time of SSDs.

The rest of this paper is organized as follows. Section II discusses the RAID architecture. Section III discusses related works and theirs problems. Section IV describes the proposed parity check and data cache management method and its operations. Section V discusses the experimental results. Section VI presents the conclusion.

II. RAID Architecture

The RAID technique is commonly used in workstations and data centers. It not only improves the I/O performance because of the parallel data access scheme but also ensures the data integrity by adding parity data.

RAID-0 uses block-level stripping, and data are written to different storage devices simultaneously. The RAID-0 has high-speed I/O performance, and RAID-0 can make use of the storage device's full capacity. However, RAID-0 is not adequately reliable because of the fact that no redundant data can be used for recovering data when a disk crashes. RAID-1 copies the data to two different devices at the same time. Therefore, the available capacity is only 50% of the total capacity. Thus, the cost of RAID-1 is considerably high.

RAID-2 uses bit-level stripping and error collection by the Hamming code. If one bit is wrong, that bit can be recovered. However, the hardware logic scheme of error collection is complicated. RAID-3, RAID-4, and RAID-5 use extra storage devices to store parities so that they can tolerate the failure of a storage device. RAID-3 uses byte-level stripping with parities. Both RAID-4 and RAID-5 use block-level stripping with parities. The minimum number of storage devices in RAID-3, RAID-4, and RAID-5 is 3. Among these three storage devices, two are responsible for storing data and the third is responsible for storing the parity data in RAID-3, RAID-4, and RAID-5.

RAID-3 and RAID-4 dispose the parity to a fixed storage device. However, when new data are written to the RAID system, the related parity must be calculated and updated. Therefore, the storage device that stores parities has a large number of write operations, and this device is extremely busy all the time. Since each block of a SSD has a limited number of erase times, the storage device that stores parities will crash in a short time.

RAID-5 disposes the parity into every storage device so that the parity write operations are separated into different storage devices. The storage capacity utilization and performance in RAID-5 are acceptable. Therefore, we adopt the RAID-5 architecture because of the above-mentioned considerations.

The bottleneck of RAID-5 is the frequent parity update. For example, the parity P_0 is generated by $D_0 \oplus D_1 \oplus D_2 \oplus D_3$, as shown in Fig. 2. The symbol "$\oplus$" represents the exclusive-OR operator. When new data are written, for example, D_1, the parity P_0 must be updated. Therefore irrespective of the amount of data that is updated in the same stripe, the parity must be computed again and written to the storage device. The cost of parity generation includes read operations that read old data from the storage device and a write operation.

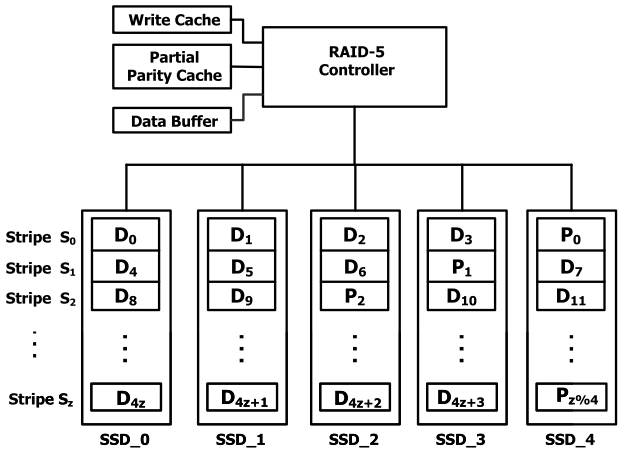

Fig. 2. Proposed 4 + 1 RAID-5 architecture.

This paper represents RAID-5 as $n + 1$, where n represents the number of data storage devices and "1" denotes the parity storage. For example, a 4 + 1 RAID-5 architecture is shown in Fig. 2.

III. Related Works

Some earlier works have used SSDs and HDDs to construct a hybrid RAID system [13]–[15]. For example, the hybrid parity-based disk array (HPDA) [13] uses SSDs to store data and two HDDs to store parities and to be a write buffer. The SSDs and an HDD are constructed by using the RAID-4 architecture. The remaining space of the parity disk HDD and another HDD are constructed by using RAID-1 as a write buffer. If the write requests are sequential, these requests are written to the RAID-4 directly. In contrast, if the requests are random access, these requests are written to the write buffer. When the I/O is idle, the requests in the write buffer are written back to the RAID-4. The HPDA [13] uses HDDs to solve the frequent parity update problem to the parity disk of the RAID-4. However, the I/O processing speed of HDDs is less than that of SSDs, and thus the life-time of the SSDs is extended, but the RAID performance is degraded.

When SSDs are directly disposed to the RAID architecture, there are a number of problems, as discussed in [16]–[18]. In the case of the RAID-4 architecture, [16] finds that parity disks have higher write times than the other disks. The experimental results show that the write operations concentrate on the parity disk, and hence their solution uses an HDD to replace the SSD parity disk.

In the case of the RAID-5 architecture, [16] proposes a wear leveling scheme that places the parity data dynamically and creates a k-bit map table for recording the number of parity write times. When one disk's write time is larger than a specific value, the wear leveling scheme exchanges those parity data with the other parity data that have lower write times. The wear leveling scheme balance only the write times of the SSDs, but the root cause of the frequent parity update problem is not solved.

To solve the frequent parity data update problem in the RAID-5 architecture, previous researches [17], [18] add a parity buffer to reduce the parity data write times. This method is called the "delay parity update scheme." The parity data are kept in the parity buffer until certain specific conditions met. The delay parity update scheme can definitely reduce the parity write times. Both [17] and [18] use the delay parity update scheme. They reduce the parity data write times, but the parity generation overhead is not considered.

The PPC scheme [10] also adopts the delay parity update scheme, but it generates a partial parity and stores it into the cache. When the cache is full, the partial parity must be rebuilt to the full parity and written to the SSD. However, the parity write times are almost the same as those of the other approaches with parity buffers. The write operation takes a considerable number of cycles, as shown in Table I. Thus if the write times for parity data can be reduced, the entire RAID performance can be significantly improved. Moreover, when the PPC scheme updates the partial parity, it needs to read the old data from the storage device, and thus the overhead for the partial parity update should be reduced.

IV. Efficient Buffer Management

A. Overall Architecture

The proposed 4 + 1 SSD-based RAID-5 architecture is shown in Fig. 2. The host system sends read or write requests to the storage interface, and the RAID-5 controller handles the write cache and distributes data access to the SSDs. The write cache holds the data from the host system, and a nonvolatile random access memory (NVRAM) is used as the write cache. The PPC stores the partial parities; it also uses the NVRAM to avoid losing data in the case of sudden power failures. A data buffer is used for reducing the costs of the partial parity update. The stored data in the data buffer can be reloaded from the SSDs, and thus a static random access memory (SRAM) or a DRAM can be used for implementing this buffer.

The concept of partial parity was proposed in [10]. A partial parity contains only a part of the full parity. The PPC can help merge parity write operations, e.g., if there are P_0 to P_{16} partial parities in the cache. The new data written to the stripes S_0–S_{16} can merge their parity write operations in the PPC. Thus, the number of write operations to the SSDs can be significantly reduced.

In this paper, we propose an efficient buffer management method to avoid the PPC being full. When the PPC is full, one of the partial parities must be selected and written back to the SSD. However, the RAID controller needs to read the associated data from the SSDs to build the full parity. In addition, the parity write operation takes a considerable number of cycles. Therefore, the proposed efficient buffer management method can reduce the costs of parity updates.

B. Structure of Each Buffer and Cache

Fig. 3 shows the data structure of the PPC, data buffer, and write cache in the proposed RAID-5 controller. The variable m denotes the number of entries in the PPC and the data buffer, and w indicates the number of entries in the write cache. T represents the total stripe number in the RAID-5, and n denotes the number of storage devices in the $n + 1$

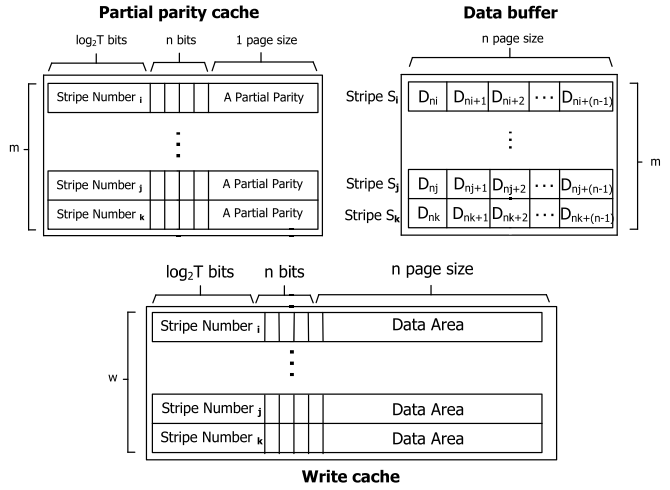

Fig. 3. PPC, data buffer, and write cache.

RAID-5 architecture. The data size of a partial parity is one page. The n-bit field represents the data associated with the partial parity. In addition, the associated data are stored in the data buffer. For example, a 4-bit binary value "1100" for the stripe number (S_0) means that the partial parity (P_0) is generated with D_0 and D_1. In addition, D_0 and D_1 are stored in the data buffer.

The data structure of the write cache is also shown in Fig. 3. The data structure of the write cache is similar to that of the PPC. The major difference is that the write cache contains data values in each entry. The variable n denotes the number of storage devices in the RAID. The size of the data area is $n \times$ the page size.

C. Comparison With PPC

Fig. 4 shows the operation flowchart proposed by the PPC [10]. The host transmits the data to the RAID controller. The RAID controller determines whether the write cache is free or not. If the write cache is free, the data can be cached in the write cache, and from the perspective of the host system, the write operation is complete. However, if the write buffer is full, the RAID controller must select the victim stripe from the write cache and write the data back to the storage device.

The selection of the victim stripe involves finding out which stripe in the write cache contains the maximum amount of data. When a stripe contains a considerable amount of data, these data can be written to the SSDs in parallel. In addition, the data in the selected stripe are used for generating a partial parity.

After generating the partial parity, the controller checks whether the PPC is free or not. If the PPC is free, the partial parity is written to the PPC, and then the data in the selected stripe are written to the storage devices. When the data are removed from the write cache, the write cache releases some free spaces.

If the PPC is full, the RAID controller selects the victim partial parity by using the least recently used (LRU) algorithm. Then, the controller rebuilds the full parity of the selected partial parity. This operation may need to read the data that

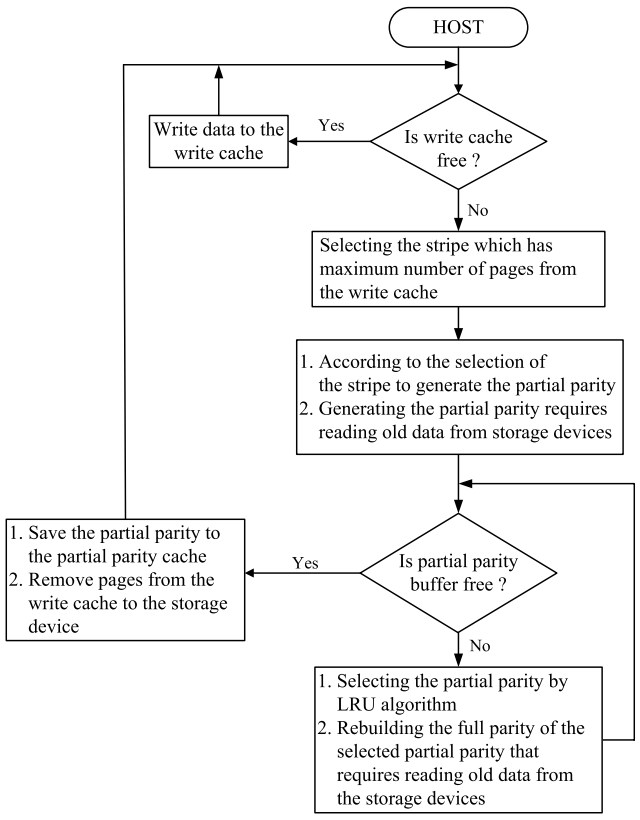

Fig. 4. Flowchart of PPC [10].

are not a part of the selected partial parity from the SSDs, and then the PPC can release some free spaces.

Fig. 5 shows the flowchart of the proposed efficient buffer management method. The major difference between the PPC [10] and the proposed method is in the selection of the victim stripe. The victim stripe selection rule in the proposed method is that, if the stripe in the write cache contains the maximum number of pages and its partial parity can be merged with the existing partial parity, this stripe will be selected first. Otherwise, we follow the selection rule of PPC [10]. The proposed method can avoid the partial PPC from being full often. When the PPC is full, the RAID controller needs to compute the full parity and write back the parity data to the storage devices; thus, the overhead is significantly large.

Moreover, we use a data buffer to reduce the cost of the partial parity update. In PPC [10], the RAID controller requires reading old data from the storage devices for updating the partial parity. In the proposed method, these old data are stored in the data buffer, and thus the number of read operations from the SSDs can be significantly reduced, and the I/O processing speed of the RAID system can be further improved.

D. Operation of the Proposed Method

The operations of the PPC, the write cache, and the data buffer are discussed in this section. The write cache stores the data from the host system. When the write cache is full, the RAID controller selects the victim stripe from the write cache. For example, Fig. 6 shows that we select the stripe S_0 to be

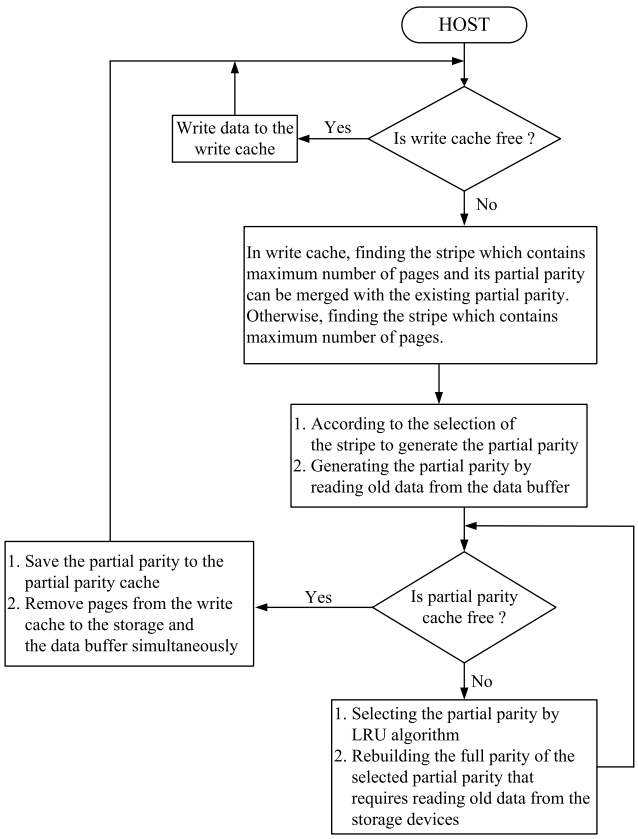

Fig. 5. Flowchart of the proposed scheme.

a victim stripe because S_0 contains the maximum amount of data (i.e., three pages). The other stripes S_5 and S_8 are still kept in the write cache to incorporate more data.

When the stripe S_0 is selected, the RAID controller performs the following operations:

1) generating the partial parity P_0 by using D_0, D_2, and D_3 and storing the ingredient bit to the PPC;
2) writing D_0, D_2, and D_3 to the corresponding locations of the data buffer;
3) writing D_0, D_2, and D_3 to the corresponding locations of the SSDs.

Fig. 6 shows the condition when the system is reset; thus there is no partial parity in the PPC, and the RAID controller directly creates a partial parity P_0. The partial parity P_0 is held in the PPC and does not write back to the SSD. When P_0 is still in the PPC, there is no further full parity update for writing D_0, D_1, D_2, and D_3. Therefore, the PPC can help reduce the full parity write times in the RAID-5 architecture. The data buffer always stores the most recent data, and the new data in the selected stripe directly overwrite the data in the data buffer.

Both the PPC and the write cache do not have sufficient free space as the host system keeps writing data to the RAID system. Then, they perform frequent selections of victim stripes and many partial parity updates. Fig. 7 shows the merging of the partial parity. In this example, there are many stripes stored in the write buffer that contain three pages of data. Further, we assume that there is no free space in the PPC.

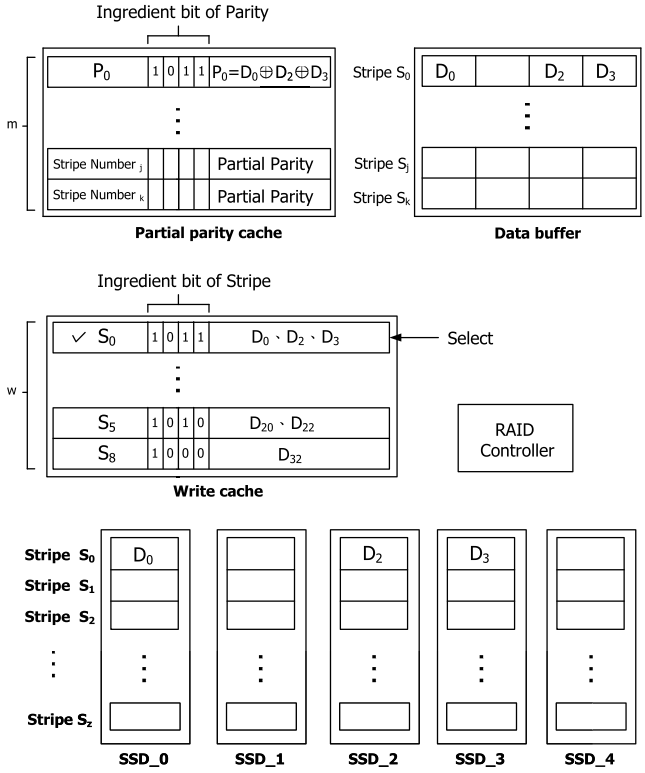

Fig. 6. Operation of the proposed scheme.

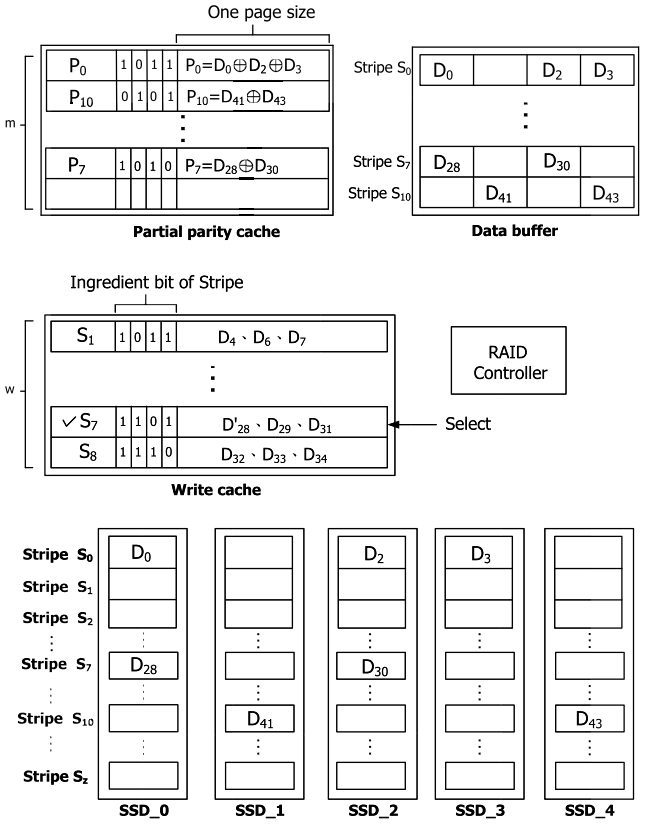

Fig. 7. Operation of computing partial parity.

In PPC [10], the RAID controller randomly selects any stripe that contains the maximum amount of data to be a victim stripe. If S_1 is selected as a victim stripe, since there is no

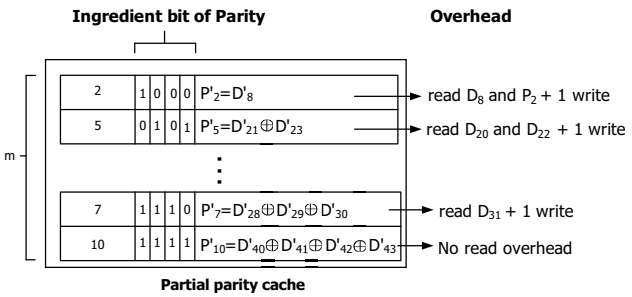

Fig. 8. Overhead of rebuilding the full parity.

free space in the PPC, the RAID controller needs to remove a partial parity from the PPC by using the LRU algorithm in order to release one free space for the new partial parity P_1. If the RAID controller decides to remove the partial parity P_7 from the PPC, the full parity generation costs for P_7 are two read operations and one write operation to the SSDs. The two read operations are performed to read D_{29} and D_{31} from the SSDs because the partial parity P_7 is generated by D_{28} and D_{30}. The one write operation is performed for writing back the generated full parity to the SSD.

Therefore, in this example, the proposed efficient buffer management method chooses S_7 as the victim stripe in order to reduce the cost of the parity update since the related partial parity P_7 already exists in the PPC. Subsequently, the RAID controller performs the following operations:

1) computing the new partial parity P_7' as follows: $P_7' = P_7 \oplus D_{28} \oplus D_{28}' \oplus D_{29} \oplus D_{31}$ (D_{28} from the data buffer);
2) writing D_{28}', D_{29}, and D_{31} to the data buffer (D_{28} will be replaced by D_{28}');
3) writing D_{28}', D_{29}, and D_{31} to SSDs (in SSD_0, D_{28} will be overwritten by D_{28}').

The new partial parity P_7' replaces the old partial parity P_7 and does not occupy a new space in the PPC. Therefore, we can reduce the number of full parity generation operations and the full parity write times to the SSDs. In addition, when the RAID controller computes the new partial parity, D_{28} can be obtained from the data buffer. Thus the proposed method reduces not only the parity write times to the SSDs but also the read times from the SSDs.

E. Overhead of Writing Back Full Parity

Fig. 8 shows the overhead of rebuilding the full parity in all possible situations. There are two methods to rebuild the full parity as follows.

Method 1: Reading the corresponding old data and the old full parity from the SSDs to rebuild the new full parity.

Method 2: Reading the data that are not a part of the partial parity from the SSDs to rebuild the new full parity.

When the partial parity P_2' needs to be written back to the SDDs, the RAID controller adopts method 1 to rebuild the full parity rather than method 2. The reason for this preference is that method 2 has three read costs (reading D_9, D_{10}, and D_{11}), which are more than the costs of using method 1. The new full parity P_2 can be computed as $P_2' \oplus D_8 \oplus P_2$.

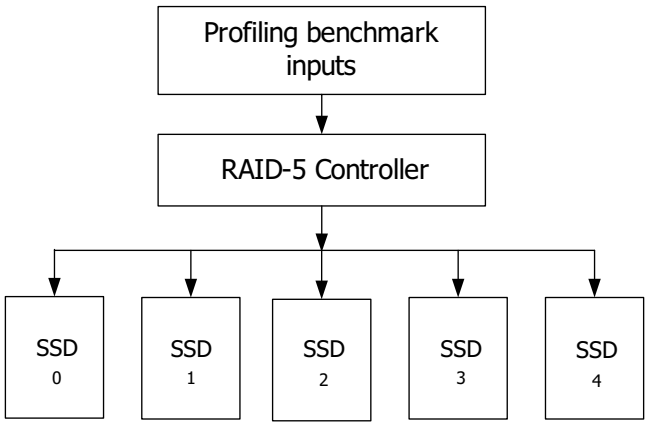

Fig. 9. 4 + 1 RAID-5 simulation environment.

The RAID controller will determine which method is more efficient. When the partial parity P_5' and P_7' must be written back to the SSDs, the RAID controller adopts method 2 to rebuild the full parity. The new full parity P_5 can be computed as $P_5' \oplus D_{20} \oplus D_{22}$, and the new full parity P_7 can be computed as $P_7' \oplus D_{31}$. When the RAID controller needs to rebuild the new full parity P_{10}, there is no read overhead since the partial parity is also the full parity.

V. Experimental Results

A. Experiment Preparation

The RAID-5 controller with the proposed efficient buffer management scheme was verified by the Socle Technology Corporation MDK-3D development board. The CPU is ARM1176JZF and the frequency is up to 1 GHz. The advanced microcontroller bus architecture advanced high-performance bus frequency is up to 200 MHz and it supports the NOR-flash/NAND-flash/DDR2 memories. We implement the RAID-5 controller and the proposed efficient buffer management scheme with the field-programmable gate array. In addition, we set up the simulation environment to evaluate the performance of the proposed RAID-5 controller as shown in Fig. 9. The entire system consists of a RAID-5 controller and SSDs [9].

The RAID-5 controller accepts the inputs from the benchmark profiling results. The stripe number and the SSD number are generated by dividing the logical address by the total number of data storage devices; the quotient and the remainder are used as the stripe number and the SSD number, respectively. For example, if the RAID system is 4 + 1 RAID-5 architecture and the logical address is 2045, 2045 is divided by 4, and the quotient and the remainder are 511 and 1, respectively. Therefore, the data with logical address 2045 are written to SSD_1 at stripe S_{511}.

The RAID controller also manages the write cache, PPC, and the data buffer. We profile two benchmarks, Iozone [19] and Postmark [20], as the inputs. On the basis of the profiling results, we use Iozone and Postmark for the sequential writing tests and the random writing tests, respectively. The reason for us to choose these two benchmarks is due to the fact that most applications contain mixing sequential writes and random writes. As a result, these two benchmarks can test the

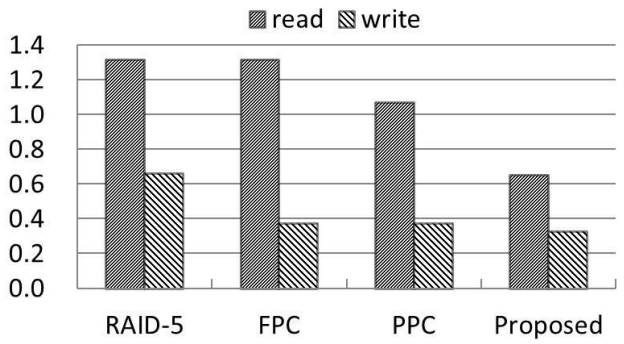

Fig. 10. Average write request overhead for generating the parity (Postmark).

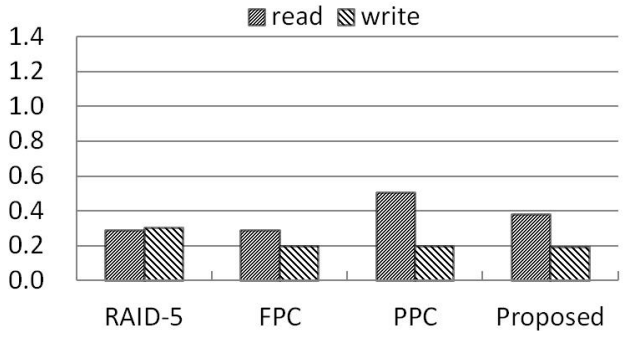

Fig. 11. Average write request overhead for generating the parity (Iozone).

best and worst performance of the proposed efficient buffer management scheme.

In the simulation environment, the page size of the SSD was 2 kB, and the size of the write cache was 16 kB. The size of the data buffer and that of the PPC are 16 kB each. Further, we rebuild four types of RAID architectures for comparison as follows.

1) RAID-5 (only has write cache).
2) FPC (RAID-5 with full parity cache).
3) PPC (RAID-5 with PPC [10]).
4) Proposed (RAID-5 with proposed PPC and data buffer).

RAID-5 is the conventional RAID-5 architecture with a write cache. FPC uses a full parity cache to reduce the parity write times. In Figs. 10, 11, 13, 14, and 16–18, we show the normalization of the number of read and write times to the SSDs by the total input of 4000 write requests.

B. Overhead of Parity Generation

Fig. 10 shows the average write request overhead for generating the parity with Postmark benchmark inputs. The conventional RAID-5 scheme does not have a parity cache, and hence the number of read operations and that of write operations for generating the parity are the highest. When the write cache in the conventional RAID-5 scheme is full, the RAID controller selects the victim stripe and writes back data to the SSDs. According to the selection of the stripe, the RAID controller builds the associate full parity and directly writes back the full parity to the SSD. When the RAID controller generates the full parity, it needs to read the remaining data of the victim stripe, and these read operations are the overhead for the full parity generation; hence the number of read operations is also the highest in the conventional RAID-5 scheme. Therefore, it is not suitable to directly add SSDs to the conventional RAID-5 architecture.

The FPC scheme has a full parity cache. The main difference between the FPC and RAID-5 is that the full parity remains in the parity cache until the parity cache is full. Therefore, some full parities can be merged in this cache, and the write times for the parity can be decreased. However, the read times for the parity generation are almost the same as those in the conventional RAID-5 scheme. The reason for this is that the full parity generation in FPC is the same as that in the conventional RAID-5 scheme.

The PPC scheme adopts the partial parity scheme, and this can reduce the read times for the parity generation. The write times of the parity are almost the same as those in the FPC scheme. We can reach two conclusions from the previously mentioned experimental results.

1) The parity cache can reduce the write times of the parity data.
2) The partial parity scheme can decrease the read times for generating the full parity.

The proposed scheme using an efficient buffer management method with a data buffer can reduce both the read times and the write times to the SSDs. The normalized write times for the parity generation of RAID-5, FPC, PPC, and the proposed scheme are 0.66, 0.37, 0.37, and 0.32, respectively. The proposed scheme decreases the number of write times for the parity as compared to PPC and RAID-5 by 13% and 51%, respectively. The normalized read times for the parity generation of RAID-5, FPC, PPC, and the proposed scheme are 1.31, 1.31, 1.068, and 0.65, respectively. The proposed scheme decreases the number of read times for parity generation as compared to PPC and RAID-5 by 39% and 50%, respectively.

Fig. 11 shows the simulation results for the Iozone benchmark. The normalized write times for the parity generation of RAID-5, FPC, PPC, and the proposed scheme are 0.3, 0.19, 0.19, and 0.18, respectively. The proposed scheme decreases the number of write times for the parity as compared to PPC and RAID-5 by 5% and 40%, respectively. The write times are decreased because of the use of the PPC and the efficient buffer method. The normalized read times for the parity generation of RAID-5, FPC, PPC, and the proposed scheme are 0.28, 0.28, 0.5, and 0.32, respectively. The proposed scheme can effectively decrease the number of read times for parity generation by 36% as compared to PPC.

However, the read times for parity generation in both PPC and the proposed scheme are larger than those in FPC and RAID-5. The reason for this is that the inputs by the Iozone benchmark are sequential with many file rewrite requests, and hence there are many partial parity updates.

The read times in the PPC scheme [10] include the read operations to the SSDs for the partial parity updates and the write operations to the SSDs for the full parity write back. In contrast, in the proposed scheme, the old data for the partial parity updates can be obtained from the data buffer, and thus the number of read operations to the SSDs for

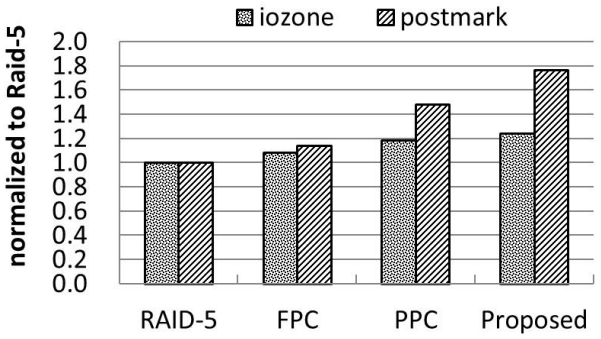

Fig. 12. RAID I/O performance.

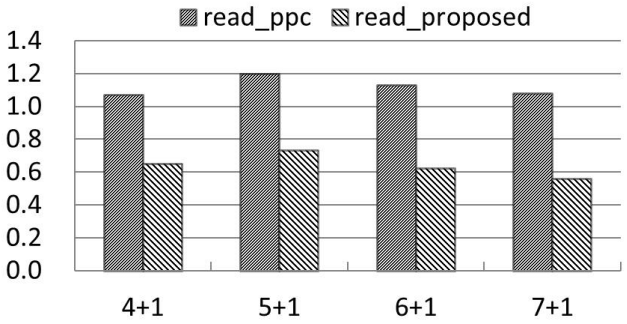

Fig. 13. Average read times for generating the parity with different stripe size (Postmark).

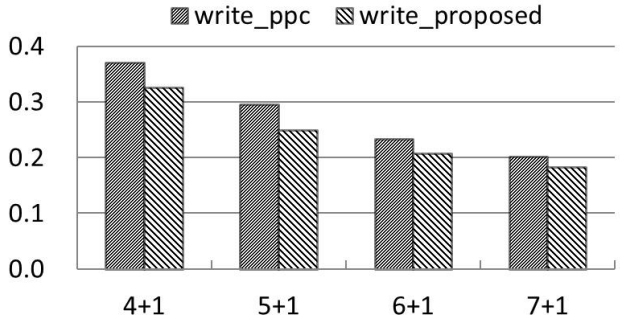

Fig. 14. Average write times for generating the parity with different stripe size (Postmark).

partial parity updates can be significantly reduced. As shown in Fig. 11, the normalized read times in PPC include 0.32 (for the partial parity updates) and 0.18 (for the full parity generation). Thus, the proposed data buffer can reduce the read times for partial parity updates with a little additional hardware cost.

C. Overall Raid I/O Performance

Fig. 12 shows the comparisons of the I/O performance for different RAID architectures. The performance is normalized to the conventional RAID-5 scheme. The RAID I/O performance with the Postmark benchmark inputs for RAID-5, FPC, PPC, and the proposed scheme are 1.0, 1.13, 1.47, and 1.76, respectively. The I/O performance of the proposed scheme is improved by 76% and 19% as compared to RAID-5 and PPC, respectively.

The detailed analysis of the total execution cycles after many write requests for the RAID system is as follows.

W_{cycle}	number of cycles spent in the write cache;
D_{cycle}	number of cycles spent in writing data to the SSDs and the data buffer;
R_{cycle}	number of cycles spent in the read operations for generating a full parity or updating a partial parity;
WP_{cycle}	number of cycles spent in writing back the full parity to the SSD;
EBF	number of cycles spent in the proposed efficient buffer management method;
LRU	number of cycles spent in the least recently used method;
T_{cycle}	total number of execution cycles.

In the proposed method, the total number of execution cycles can be expressed as follows:

$$T_{\text{cycle}} = \text{cycle} + D_{\text{cycle}} + R_{\text{cycle}} + WP_{\text{cycle}} + \text{EBF} + \text{LRU}. \quad (1)$$

In the PPC [10], the total number of execution cycles can be expressed as follows:

$$T_{\text{cycle}} = W_{\text{cycle}} + D_{\text{cycle}} + R_{\text{cycle}} + WP_{\text{cycle}} + \text{LRU}. \quad (2)$$

Finally, the total number of execution cycles for RAID-5 and FPC can be expressed as follows:

$$T_{\text{cycle}} = W_{\text{cycle}} + D_{\text{cycle}} + R_{\text{cycle}} + WP_{\text{cycle}}. \quad (3)$$

The random write requests occur more often than the sequential write requests in the RAID system. The proposed scheme handles the random write requests very well because the parity merging often takes place in the PPC with random inputs. The R_{cycle} and WP_{cycle} of the proposed scheme are smaller than those of the PPC [10].

The 4 + 1 RAID-5 I/O performance with the Iozone benchmark inputs is nearly the same for different RAID architectures because the RAID controller can easily find a stripe with four pages of data with the sequential write requests. When the RAID controller selects a stripe with four pages of data, the partial parity associated with this stripe is also considered the full parity, and thus both the PPC scheme and the proposed scheme do not need to rebuild the full parity. The R_{cycle} and WP_{cycle} in each of the RAID architectures are almost the same in this case.

D. Simulation With Difference Parameters

The average read times and write times for generating the parity with different stripe sizes are shown in Figs. 13 and 14, respectively. The write times of the proposed scheme for generating the parity are smaller than those of the PPC [10] because of the proposed efficient buffer management method, as shown in Fig. 5. When the stripe size increases, the write times for generating the parity decrease in both PPC and the proposed scheme. This is attributed to the fact that the partial parity can merge more data with a relative large stripe size. For example, the partial parity P_0 in the 4 + 1 RAID-5 system can represent D_0 to D_3. However, the partial parity P_0 in the 7 + 1 RAID-5 system can represent D_0 to D_7. This means the possibilities of merging the partial parities are increased.

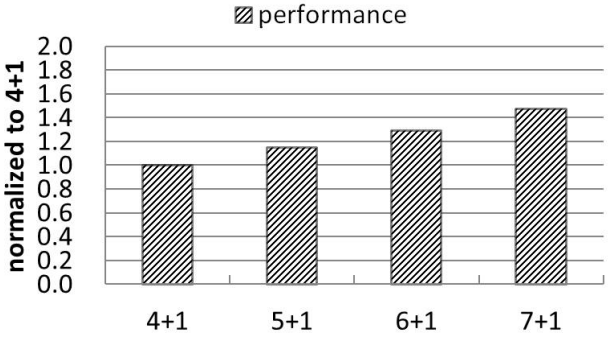

Fig. 15. RAID I/O performance with different stripe size (Postmark).

The read times for generating the parity in the proposed scheme is also smaller than in the PPC [10] with different stripe sizes. There are two reasons for the fact that the read times for generating the parity are relatively small in the proposed scheme: First, the write times for generating the parity of the proposed scheme is smaller than PPC, and thus the related read times for rebuilding the full parities are also reduced. Second, the proposed scheme adds a data buffer to store the old data for the partial parity updates, and thus the read times for generating the parity can be further reduced.

Fig. 15 shows the RAID I/O performance of the proposed scheme with different stripe sizes. The performance is normalized to the 4 + 1 RAID-5 system. The I/O performance of 4 + 1, 5 + 1, 6 + 1, and 7 + 1 RAID systems is 1.0, 1.15, 1.29, and 1.47, respectively. The read times and write times for generating the parity are decreased gradually with an increase in the stripe size as shown in Figs. 13 and 14. Further, the RAID controller can easily write data to the SSDs in parallel in the case of a relatively large stripe size. As a result, when the stripe size increases, the I/O performance also improves.

Fig. 16(a) and (b) shows the ratio of the read times of the PPC scheme [10] and the proposed scheme with the Postmark benchmark inputs. In the case of the full parity write back operations in PPC, the ratios are 0.73, 0.79, 0.64, and 0.51. In the case of the partial parity update to the SSDs in PPC, the ratios are 0.33, 0.40, 0.48, and 0.56. In the case of PPC, the ratio of the read times for the partial parity update increases gradually with an increase in the stripe size because the partial parity is associated with more data in the case of a relatively large data stripe. The proposed scheme adds a data buffer to retain the old data for the partial parity updates, and therefore there is no read operation to the SSDs for the partial parity update, as shown in Fig. 16(b).

Fig. 17(a) and (b) shows the ratio of the read times of the PPC scheme [10] and the proposed scheme with the Iozone benchmark inputs. In the case of the full parity write back operations in PPC, the ratios are 0.18, 0.64, 0.53, and 0.34. In the case of the partial parity update to the SSDs in PPC, the ratios are 0.32, 0.39, 0.47, and 0.54. The read times for the full parity generation with sequential inputs are smaller than those in the case of random writing inputs. In Fig. 17(b), we see that the total read times in the proposed scheme are reduced by adding the data buffer.

The average page size when the RAID controller selects a victim stripe from the write buffer with the Iozone and the

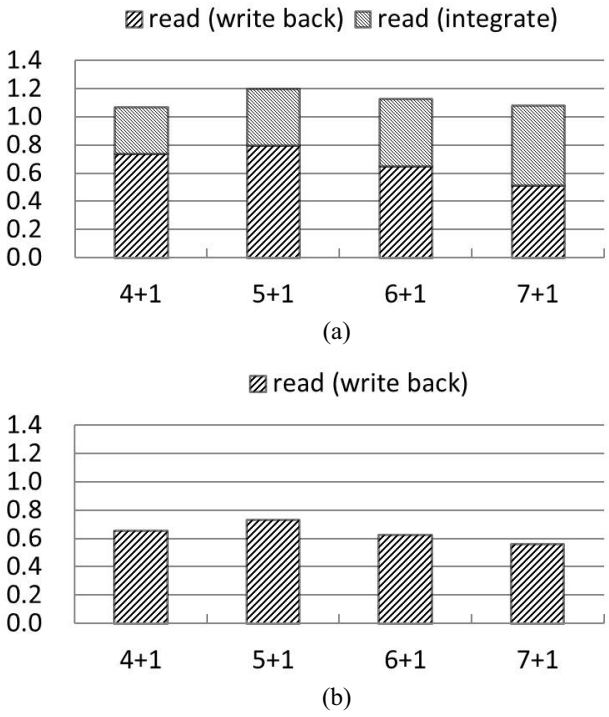

Fig. 16. Ratio of read times for generating the parity (Postmark). (a) PPC. (b) Proposed scheme.

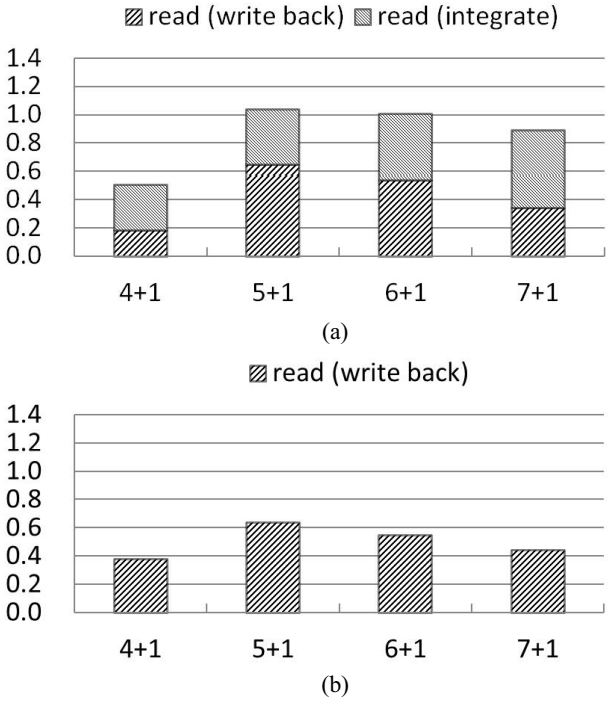

Fig. 17. Ratio of read times for generating the parity (Iozone). (a) PPC. (b) Proposed scheme.

Postmark benchmarks is 3.0 and 1.3 pages, respectively. In the case of the Iozone benchmark, the RAID controller may need a 1.0 read operation on average for generating the full parity. In the case of the Postmark benchmark, the RAID controller may need 2.7 read operations on average for generating the

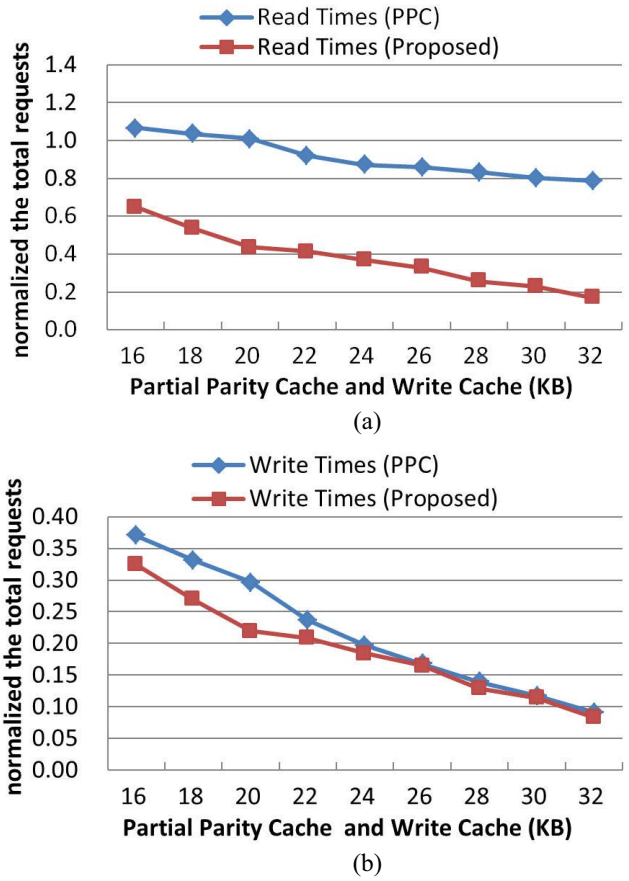

Fig. 18. Average read times and write times for generating the parity with different cache size. (a) Read times. (b) Write times.

full parity. Thus, the number of read operations in the case of the Postmark inputs is larger than that in the case of the Iozone inputs, as shown in Figs. 16 and 17.

Fig. 18(a) and (b) shows the read times and write times for generating the parity of the PPC scheme [10] and the proposed scheme with different cache sizes. Fig. 18(a) shows that the read times for generating the parity of the proposed scheme are always smaller than those in the case of PPC. The read times in the case of the proposed scheme are 0.65 and 0.16 with the PPC and the write cache size of 16 kB and 32 kB, respectively. The read times of the PPC scheme are 1.06 and 0.80 with the PPC and the write cache size of 16 kB and 32 kB, respectively. A relatively large PPC can merge the partial parities more efficiently. Moreover, the data buffer in the proposed scheme helps to reduce the read times required for generating the parity.

Fig. 18(b) shows the write times for the full parity write back operations to the SSDs of the PPC scheme [10] and the proposed scheme with different cache sizes. The proposed scheme can reduce the write times even with a relatively small cache. As a result, we can process the full parity write back operations well with limited hardware resources. In the case of a relatively large cache size, the write times can be reduced in both the PPC scheme and the proposed scheme. However, with the proposed efficient buffer management scheme, the write times of the proposed scheme will be always smaller than those of the PPC scheme.

VI. Conclusion

In this paper, we proposed a PPC and data cache management method to improve the performance of an SSD-based RAID system. There were many considerations for adding SSDs to the RAID-5 architecture because the characteristics of SSDs were different from those of traditional HDDs.

The proposed PPC with an efficient buffer management method could merge the partial parity data more efficiently. The proposed RAID controller selected the suitable victim stripe from the write cache in order to prevent the PPC from often being full. When the PPC was full, the RAID controller used the LRU algorithm to select the victim partial parity, rebuilt the full parity of the selected parity, and wrote it back to the SSDs. We also added a data buffer to reduce the partial parity update overhead. Experimental results revealed that both the number of the read operations and the number of the write operations to the SSDs for generating the parity were reduced by the proposed scheme.

Acknowledgment

The authors would like to thank their colleagues at the Silicon Sensor and System Laboratory, National Chung Cheng University, Chia-Yi, Taiwan, for engaging in many fruitful discussions, and the National Chip Implementation Center for providing the EDA tools.

References

[1] R. Ho, K. W. Mai, and M. A. Horowitz, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," in *Proc. ACM/IEEE ISCA*, Jun. 2009, pp. 279–289.

[2] C. Lee, S. H. Baek, and K. H. Park, "A hybrid flash file system based on NOR and NAND flash memories for embedded devices," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 1002–1008, Jul. 2008.

[3] L.-P. Chang, "A hybrid approach to NAND-flash-based solid-state disks," *IEEE Trans. Comput.*, vol. 59, no. 10, pp. 1337–1349, Oct. 2010.

[4] C.-C. Chung, D. Sheng, and N.-M. Hsueh, "A high-performance wear-leveling algorithm for flash memory system," *IEICE Electron. Exp.*, vol. 9, no. 24, pp. 1874–1880, Dec. 2012.

[5] A. Nishi, "Datacenter power savings through high ambient datacenter operation: CFD modeling study," in *Proc. IEEE SEMI-THERM*, Mar. 2012, pp. 104–107.

[6] Hitachi Global Storage Technologies (GST). (2010). *Solid State Drives for Enterprise Data Center Environments*, San Jose, CA, USA [Online]. Available: http://www.hgst.com/tech/techlib.nsf/techdocs/F81A37DF296938BF8625763B00048D41/$file/SSD_techbrief.pdf

[7] D. Reinsel and J. Janukowicz. (2008). *Datacenter SSDs: Solid Footing for Growth* [Online]. Available: http://www.samsung.com/us/business/semiconductor/news/downloads/210290.pdf

[8] G. Schulz. (2007). *Achieving Energy Efficiency Using Flash SSD* [Online]. Available: http://www.cristie.co.uk/uploads/media/Achieving_Energy_Efficiency_Using_SSD.pdf

[9] Hynix Corporation. (2007). *HY27UK08BGFM NAND Flash Memories*, Raunheim, Germany [Online]. Available: http://www.hynix.com/inc/pdfDownload.jsp?path=/datasheet/pdf/flash/HY27UK08BGFM%20(Rev0.0).pdf

[10] S. Im and D. Shin, "Flash-aware RAID Techniques for dependable and high-performance flash memory SSD," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 80–92, Jan. 2011.

[11] H. Kim and U. Ramachandran, "FlashLite: A user-level library to enhance durability of SSD for P2P file sharing," in *Proc. IEEE ICDCS*, Jun. 2009, pp. 534–541.

[12] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Dacis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," in *Proc. USENIX ATC*, Jun. 2008, pp. 57–70.

[13] B. Mao, H. Jiang, D. Feng, S. Wu, J. Chen, L. Zeng, and L. Tian, "HPDA: A hybrid parity-based disk array for enhanced performance and reliability," in *Proc. IEEE IPDPS*, Apr. 2010, pp. 1–12.

[14] J. Gen and Q. Yang, "I-CASH: Intelligently coupled array of SSD and HDD," in *Proc. HPCA*, Feb. 2011, pp. 278–289.

[15] Y. Liu, J. Huang, C. Xie, and Q. Cao, "RAF: A random access first cache management to improve SSD-based disk cache," in *Proc. NAS*, Jul. 2010, pp. 492–500.

[16] K. Park, D.-H. Lee, Y. Woo, G. Lee, J.-H. Lee, and D.-H. Kim, "Reliability and performance enhancement technique for SSD array storage system using RAID mechanism," in *Proc. ISCIT*, Sep. 2009, pp. 140–145.

[17] K. M. Greenan, D. D. E. Long, E. L. Miller, T. J. E. Schwarz, and A. Wildani, "Building flexible, fault-tolerant flash-based storage systems," in *Proc. HotDep*, Jun. 2009.

[18] Y. Lee, S. Jung, and Y. H. Song, "FRA: A flash-aware redundancy Array of flash storage devices," in *Proc. Hardw./Softw. Codesign Syst. Synthesis*, Oct. 2009, pp. 163–172.

[19] *Iozone File System Benchmark* [Online]. Available: http://iozone.org

[20] *Postmark File System Benchmark, Network Appliance* [Online]. Available: http://www.netapp.com

Ching-Che Chung (S'01–M'03) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1997 and 2003, respectively.

He was a Post-Doctoral Researcher with National Chiao-Tung University from 2004 to 2008, on system-on-chip design methodologies and high-speed interface circuit design. In August 2008, he joined the Faculty of the Computer Science and Information Engineering Department, National Chung Cheng University, Chia-Yi, Taiwan, where he is currently an Associate Professor. His current research interests include wireless and wireline communication systems, low-power and system-on-a-chip design technology, mixed-signal IC design and sensor circuits design, all-digital phase-locked loop, and all-digital delay-locked loop and its applications.

Hao-Hsiang Hsu received the M.S. degree in computer science and information engineering from National Chung Cheng University, Chia-Yi, Taiwan, in 2012.

He is currently a Software Engineer with the Research and Development Department, ALi Corporation, Taipei, Taiwan, on set-top box system applications. His current research interests include system-on-a-chip design methodologies and flash-based storage systems.