CR5M: A Mirroring-Powered Channel-RAID5 Architecture for An SSD

Yu Wang
Computer & Information School
Hefei University of Technology
Hefei, Anhui, China
yuwang0810@gmail.com

Wei Wang, Tao Xie
Computer Science Department
San Diego State University
San Diego, California, USA
wang@rohan.sdsu.edu,
txie@mail.sdsu.edu

Wen Pan, Yanyan Gao, Yiming Ouyang
Computer & Information School
Hefei University of Technology
Hefei, Anhui, China
{wenwen412, littlek.gao}@gmail.com
comoyym@hfut.edu.cn

***Abstract*— Manufacturers are continuously pushing NAND flash memory into smaller geometries and enforce each cell to store multiple bits in order to largely reduce its cost. Unfortunately, these scaling down techniques inherently degrade the endurance and reliability of flash memory. As a result, permanent errors such as block or die failures could occur with a higher possibility. While most transient errors like programming errors can be fixed by an ECC (error correction code) scheme, rectifying permanent errors requires a data redundancy mechanism like RAID (redundant array of independent disks) in a single SSD where multiple channels work in parallel. To enhance the reliability of a solid-state drive (SSD) while maintaining its performance, we first implement several common RAID structures in the channel level of a single SSD to understand their impact on an SSD's performance. Next, we propose a new data redundancy architecture called CR5M (*C*hannel-*R*AID*5* with *M*irroring), which can be applied to one SSD for mission-critical applications. CR5M utilizes hidden mirror chips to accelerate the performance of small writes. Finally, we conduct extensive simulations using real-world traces and synthetic benchmarks on a validated simulator to evaluate CR5M. Experimental results demonstrate that compared with CR5 (*C*hannel-*R*AID*5*) CR5M decreases mean response time by up to 25.8%. Besides, it reduces the average writes per channel by up to 23.6%.**

Keywords—flash memory; solid-state disk; SSD; RAID5

I. INTRODUCTION

NAND flash memory based solid state drives (hereafter, SSDs) possess some attractive properties such as low energy consumption, high shock resistance, and small physical size compared to traditional hard disk drives (HDDs). Thanks to the development of manufacturing process, the price of SSDs in terms of GB/$ has been rapidly decreased, which makes them an alternative to HDDs in portable and mobile computing devices such as laptops and tablets [1][6][9]. Increasing flash capacity density is an effective way to reduce its cost. Typically, there are two methods to densify a flash chip. One is to shrink flash memory cell into smaller geometries (e.g., from 45 *nm* to 20 *nm*). The other is to push each cell to store more bits (e.g., from SLC to MLC) [4][5]. Unfortunately, both methods have significant side effects on flash endurance and reliability [5][7]. For example, a typical SLC (single-level cell) can tolerate ~100k P/E (program/erase) cycles with a 1-bit ECC capacity per 512 bytes, whereas a 2-bit MLC (multi-level cell) can only survive ~10k P/E cycles with a more-than-4-bit ECC capacity per 512 bytes [3]. The decreasing endurance and reliability exert continuous pressures on maintaining data integrity and availability in SSDs [4][5][8].

A reliability-degraded SSD tends to generate more errors. There are two types of errors: transient (or soft) errors and permanent (or hard) errors. Typical transient errors include programming error and read error [4]. Block or die errors are categorized to permanent errors. Although an ECC scheme can correct both types of errors, when the number of errors exceeds its capacity a data loss will occur. Thus, a data redundant mechanism is demanded to protect data under permanent errors. A well-known such mechanism is the RAID (redundant arrays of independent disks) [22] organization, which has successfully been implemented in HDD arrays, SSD arrays, and hybrid arrays [11][12][13][15][16]. Essentially, it trades capacity loss for data reliability. RAID-5, one of the most widely used RAID organizations, distributes parities along with data and can tolerate a single disk failure [2].

Usually, a RAID architecture is applied in server-class applications where multiple HDDs or SSDs are organized in one array. Still, there are many cases where only one SSD can be deployed due to space and energy constraints yet data reliability is critical [23][25]. For example, a wireless healthcare system collects community and clinical health data and monitors patient vital signs in real time [23]. It requires a high level of data reliability as data sampled from mobile and dynamic environments are most likely irreproducible [23]. Another example is a mobile military application, which uses mobile devices to carry out intelligence and tactical operations where data loss could impact national security [25].

The internal structure of an SSD exhibits a hierarchical architecture. An SSD consists of multiple channels with each one having one or multiple chips [1]. Each channel can work in parallel just like an independent disk does. The multi-channel structure provides us with an opportunity to implement various RAID formats into a single SSD to form a channel-RAID (CR) architecture such as CR1 (Channel-RAID1), CR4 (Channel-RAID4), and CR5 (Channel-RAID5) [6][17][18]. However, simply transplanting a RAID structure at the channel level faces several challenges. Take RAID5 for example. Firstly,

978-1-4799-5671-5/14/$31.00©2014 IEEE

frequent parity updates largely burden the endurance of an SSD whose P/E cycles are already limited [3]. Besides, they could noticeably shorten an SSD's longevity because some chips that receive a larger number of writes will wear out more quickly than others. Secondly, the parity updates increase an SSD's write amplification, which leads to a poorer SSD performance.

To solve these challenges, in this paper we first implement several widely known RAID structures in the channel level of a single SSD like CR1 (*C*hannel-*R*AID*1*), CR4 (*C*hannel-*R*AID*4*), and CR5 (*C*hannel-*R*AID*5*) to fully understand their impacts on performance and reliability. Next, we propose a mirroring-powered channel-RAID5 (CR5M) architecture, which can be applied to one SSD for mission-critical applications. CR5M organizes multiple channels of an SSD in a RAID5-like manner where each channel acts as an independent disk. Each channel has one or more identical chips. In addition, a hidden mirroring chip is added to each channel. RAID5 offers a good balance between storage efficiency and I/O performance. However, its performance suffers when it serves small random writes/updates dominant workloads [2]. The reason behind this is that RAID5 has to perform more additional read operations, as more stripes have to calculate a new parity. Moreover, small random writes/updates increase data copying overhead during garbage collection operations so that an SSD's performance and reliability will be further deceased. Therefore, CR5M attaches an extra chip that serves as a mirroring device under each channel of an SSD to mitigate the problems of RAID5. When a small random write/update request arrives, CR5M first dispatches it to its destination channel. And then, CR5M concurrently writes it onto a particular data chip and the mirroring chip without updating the parity if both chips are ready to serve. Clearly, the newly written data is protected by mirroring, whereas other data in the same stripe are still protected by the present parity. The parity will be finally updated when the mirroring chip is not ready to serve a new update or its available capacity reaches a predefined threshold. In this way, the overall performance can be boosted by postponing parity updates due to small random writes/updates.

SSDs employ FTL (flash translation layer) to map a logical block address to a physical flash address [19][20]. The FTL hides the internal flash memory organization, IO operations, and data placement from the operating system. We implement CR1, CR4, CR5, and CR5M architectures within a page-mapping FTL presented in a validated SSD simulator SSDsim [6] without introducing an extra hardware cost. Our experimental results demonstrate that in terms of mean response time CR5M outperforms CR5 by up to 25.8%. Compared with CR4, CR5M can achieve a performance gain up to 33.4%. Besides, CR5M reduces the average writes per channel in a range of 4.5% to 23.6% compared with CR5.

The rest of the paper is organized as follows. Related work and motivation will be presented in Section II. In Section III, a discussion of how a channel-RAID5 organization is employed in a single SSD will be first provided, and then the implementation of CR5M is presented. The performance and reliability of CR5M is evaluated using a validated simulator under real-world and synthetic traces in Section IV. Section V concludes this paper.

II. Related Work and Motivation

A. SSD Basics

Due to the absence of moving parts, HDD's long seek time and rotational latency are avoided by SSDs. The flash memory part of an SSD is composed by an array of identical chips. Several chips share a channel, which connects them to the flash controller. All chips within one channel have separate chip enable and read/busy control signals [3]. Each chip consists of multiple dies and each die has its own internal ready/busy signal. Further, each die contains multiple planes with each having thousands of blocks and one or two data/cache register as an I/O buffer. Each block typically has 64 or 128 pages. The size of one page normally varies from 2 KB to 16 KB [2]. Each page contains a spare area used for error correction and metadata. There are three basic operations in flash: read, write, and erase. While read and write are performed at page-level, erase can be carried out only at block-level. Each block must be erased before it can be written, which is a characteristic known as *erase-before-write*. An erase operation is a time-consuming operation compared with read and write. Each block can only sustain a limited number of erase operations. The *erase-before-write* is hidden by using an *out-of-place* update method: first, the update data is written to an erased page; next, the page that contains the old data is invalidated; finally, the virtual-to-physical address mapping table is modified to reflect this change [1].

Recently, manufacturers are aggressively pushing flash memory into smaller geometries to further decrease the cost per gigabyte. At the same time, flash memory are moving to store more bits per cell to further increase the storage density. Compared with SLC, MLC and triple-level cell (TLC) has become the dominant form of NAND flash and constitutes about 90% of the flash parts shipped. However, these technologies negatively impact the endurance and reliability of flash memory [4]. For example, a SLC can tolerate 10k program/erase (P/E) cycles, whereas a 2-bit MLC can only survive for 3k P/E cycles for 30-40 nm technology generations. The available P/E cycles will further decrease in the future as flash cells keep scaling down in size and each cell stores more than 2 bits. As a result, SSDs wear out more quickly, especially for write-intensive applications.

B. Flash Memory Errors

In addition to a reduced longevity, the manufacturing process scaling technology also leads to a degraded SSD reliability. As each bit cell gets smaller, fewer electrons can be trapped in the floating gate, which result in more errors [4]. At the same time, manufacturers are moving past two-bit MLC to three-bit TLC to further increase storage density [5]. However, as the number of bits stored per cell increases, bit values are represented by smaller voltage ranges. A smaller voltage range generates an uncertainty in the value stored. Consequently, the flash raw bit error rate (RBER) increases [5].

Normally, there are two types of flash errors: transient errors and permanent errors. Transient errors can be further classified into four groups from the controller's point of view: erase error, programming interference error, retention error, and read error [4]. Ideally, all transient errors would be corrected by an ECC algorithm. In reality, the ECC scheme

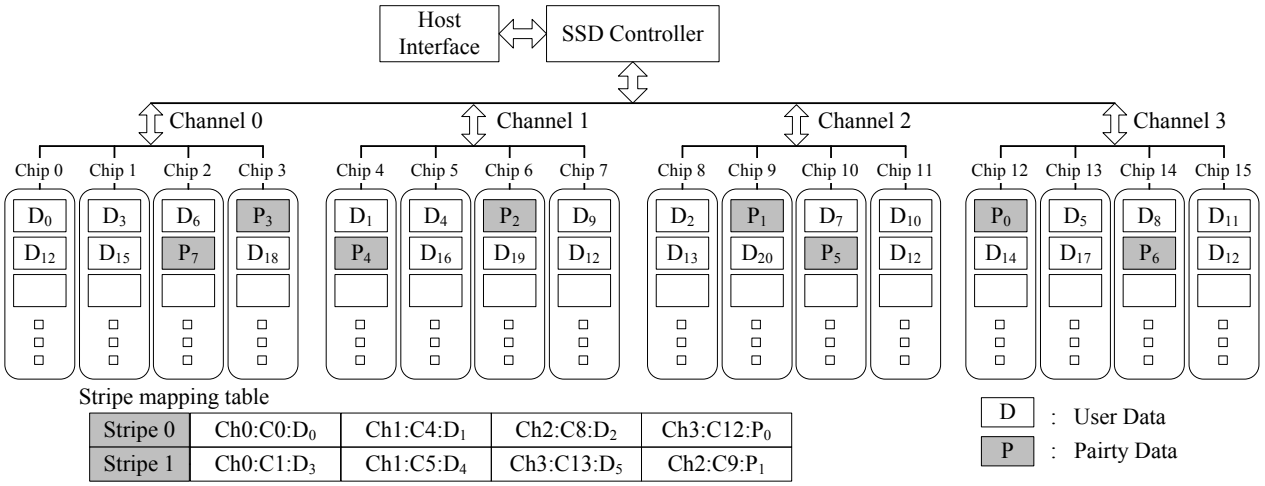

Fig. 1. The architecture of CR5 SSD.

only protects against a range of errors. Typically, ECC can detect two bit errors and correct one bit error per 256 to 512 bytes [4]. Errors beyond that range may be unrecoverable. On the other hand, permanent errors include word line errors, and block or chip errors. However, ECC schemes are incapable of correcting these permanent errors. Thus, in this circumstance a data redundancy mechanism like RAID5 has to be utilized to protect data. Apparently, a data redundancy scheme trades capacity for an improved performance and reliability.

C. Motivation

SSDs are increasingly used in RAID arrays to replace or cooperate with traditional HDDs. These SSD arrays or SSD-HDD hybrid arrays focus on enhancing the RAID controller including parity updating schemes and wear-leveling algorithms in order to improve the performance and lifetime of the arrays [8][9][10][12][13][14][18][24]. Im and Shin proposed a scheme using the partial parity technique to reduce the number of read operations required to calculate a parity [8]. Similarly, Lee *et al.* developed a new technique called FRA (Flash aware Redundancy Array). In this technique, parity updates are postponed so that they are not included in the critical path of read and write operations [10]. To enhance the reliability of an SSD array, Kadav *et al*. presented Diff-RAID, a new RAID variant that distributes parity unevenly across SSDs to create age disparities within arrays [9].

However, the traditional multi-device RAID structure is impractical to portable and mobile computing devices where only a single SSD can be deployed. Fortunately, as Hu *et al.* pointed out, an SSD contains inherent internal parallelism at multiple levels [6][15]. This provides an opportunity to migrate a RAID structure into a single SSD. After a study on the parallelism of different levels within an SSD, we argue that implementing a RAID architecture in the channel level is most appropriate because channel level can improve performance more effectively than all the other levels (e.g., chip level, plane level). The high data reliability requirements imposed by mission-critical mobile applications as well as our investigations on the internal structure of an SSD motivate us to develop a channel-RAID architecture to improve the performance and reliability of a single SSD.

III. THE CR5M ARCHITECTURE

In this section, we first introduce how the RAID5 organization is implemented within a single SSD. Next, the design and implementation of CR5M architecture are presented.

A. Architecture of CR5

An SSD provides a multi-level parallelism (e.g., channel, chip, and die level). Hu *et al.* [6] suggested that an optimal priority order of parallelism in a single SSD should be: channel-level, die-level, plane-level, and chip-level. Considering the fact that flash memory chip is the smallest replaceable unit in SSD and multiple chips are grouped together by sharing one channel, we implement the RAID architecture on the channel level. Fig. 1 shows an example of a four-channel SSD with each channel consisting of four chips. When channel-RAID5 is adopted user data will be written in a stripe unit. In the above example, a stripe contains 3 pieces of user data and 1 parity data. For example, user data $D_0 \sim D_2$ and parity P_0 comprise stripe 0. More generally, the stripe index j of data D_i can be calculated as $j = \lfloor i / (N - 1) \rfloor$ while i is the index of D_i and N is the number of channels. The parity of the stripe j, P_j, is equal to $D_{j \cdot (N-1)} \oplus D_{j \cdot (N-1)+1} \oplus \cdots \oplus D_{j \cdot N - 1}$ where $\oplus$ is the exclusive-OR (XOR) operation. The striping size of CR5 is adjusted to $(N - 1)$ page size. The reason of choosing $(N-1)$-page striping size is two-fold. Firstly, a page is the smallest granularity of read/write operation so that one-page size user data simplifies the read/write operations within CR5 SSD and its data management. Secondly, for multi-page requests each of them can be divided into multiple one-page requests, which can fully utilize parallelism by distributing these requests into different channels.

Semi-Dynamic Allocation Scheme. Traditionally, the layout of each stripe in RAID5 structure is fixed, which is also referred to as static allocation scheme. The static allocation scheme is efficient because computing the address of each

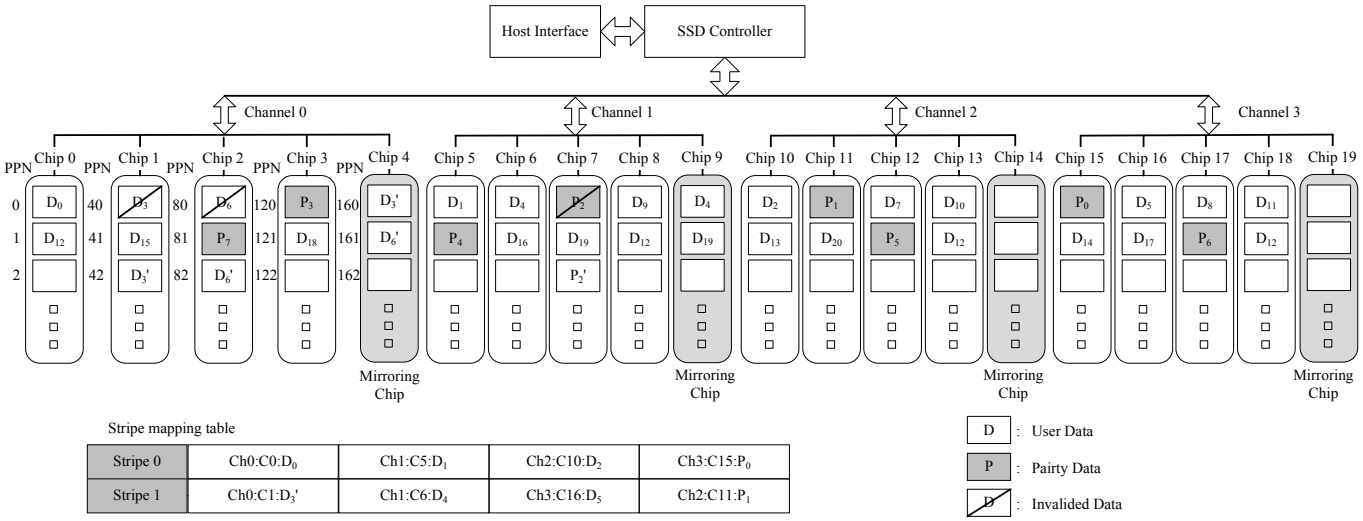

Fig. 2. The architecture of CR5M SSD.

```
1.  Input: N, j, i, type(request)
2.  Output: channel_addr, chip_addr, page_addr
3.  IF type(request) == parity write THEN
4.      channel_addr = (N – 1) – (j % N);
5.  ELSE IF type(request) == user data write THEN
6.      parity_channel = (N – 1) – (j % N);
7.      channel_addr = i % (N – 1);
8.      IF (channel_addr >= parity_channel) THEN
9.          channel_addr++;
10.     END IF
11. END IF
12. chip_addr = find_next_available(channel_addr);
13. page_addr = find_next_available(chip_addr);
```

Fig. 3. The SDA algorithm.

piece of data (i.e., user data and parity) is simple. In SSD, each channel consists of multiple chips. These chips share data and command buses and can be used in an interleaving way. The overall performance of an SSD can be noticeably improved if the interleaving is fully utilized [6]. Instead of using static allocation scheme, in CR5 we adopt a flexible allocation scheme called semi-dynamic allocation (SDA) to fully utilize interleaving between chips, which under the same channel. There are two steps in the SDA scheme. First, SDA calculates the channel address of each request using a predefined equation (i.e., static allocation). User data and parity are treated in a different way (see Fig. 3). Next, the chip address and page address is dynamically allocated by detecting the availability of each chip on the channel address, which is determined by step 1. Fig. 3 illustrates the SDA algorithm. *N*, *j*, and *i* represent the number of channels, stripe index, and data index within one stripe, respectively.

When a request arrives, the SSD controller first splits it into multiple one-page size sub-requests. And then they are grouped according to the stripe number of the data they accessed. There are two types of write. One is full-stripe write, which handles a group of sub-requests across the whole stripe. Its parity data can be directly computed. The other one is partial-stripe write. It contains sub-requests that only access a part of a stripe. To calculate its parity several pre-read operations must be performed. Usually, two alternative methods are used for updating parity data in a partial-stripe request: RMW (Read-Modify-Write) and RCW (Read-Reconstruct-Write). RMW reads the old data of the updates and its associated parity. Thus, the number of pre-read operations of RMW equals to the number of updates plus one (i.e., parity read). RCW, on the contrary, reads the rest part of the stripe (i.e., the data that are not going to be updated), so the number of pre-read operations of RCW is equal to the number of data pages in a stripe minus the number of updated pages. To reducing the overhead of the pre-read operations, the method whose pre-read operation number is less will be selected. If they are equal, RCW is adopted because it does not depend on the parity information so that the probability of data errors becomes lower [2].

Limitation of CR5 SSD. There are several limitations to this approach. First, the chip in which the parity page resides is more prone to wear out for it has to be written more frequently. This leads to a decreased lifetime of an SSD. Second, additional read operations must be applied in partial-stripe requests to calculate new parity whether RCW or RMW is employed. The additional read operations can significantly increase the mean response time of storage system, especially when the majority writes are random small updates. Furthermore, during RCW or RMW procedure the parity data cannot be written until all the read operations are carried out, leaving open a window of vulnerability. Hence, the CR5 architecture is not suitable for flash based SSDs.

B. Architecture of CR5M

A CR5M is proposed to combat the shortcomings in traditional RAID5 organization. The key feature of CR5M is that an extra chip is introduced to each channel, which is transparent to users and does not contribute for the total SSD capacity. These chips serve as a mirroring chip. However, they only store mirroring data for small random updates in partial stripe. Fig. 2 shows an example architecture of CR5M SSD.

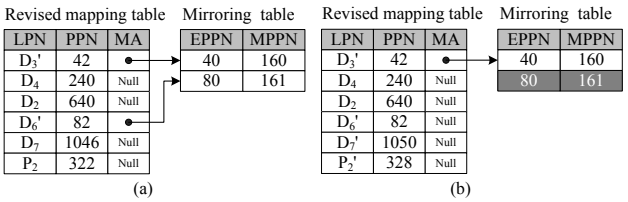

Fig. 4. (a) Revised mapping table in CR5M and (b) mapping table after an expired data reclaim process.

There are 5 chips on each channel. The last one is the mirroring chip (e.g., chips 4, 9, 14, and 19). In addition to RMW and RCW, CR5M provides another special procedure called mirroring write (MW), which can only be invoked in a partial stripe update. MW concurrently writes both the original update and a copy of it onto its destination chip and the mirroring chip, respectively. As chips on the same channel can work in an interleaving way, the overhead of an MW operation is equivalent to a write operation.

Let us using an example shown in Fig. 2 to clarify how the CR5M and MW operation works. Assuming the entire SSD is idle at this point and a partial stripe update that contains only one page data D_3' arrives. As the MW operation is triggered two identical pieces of D_3' are written onto chip 1 and mirroring chip (i.e., chip 4) simultaneously. Different to the conventional data and parity updating procedure, MW will not invalid the old data D_3 on chip 1 and calculate the new parity for this stripe. To reclaim the obsolete data and parity an expired data reclaim process will be carried out when MW is disabled. There are two scenarios that MW will be disabled. The first one is at the time when the available capacity of mirroring chip is lower than a predefined threshold. The second one is the time when the mirroring chip is busy. As the mirroring data and obsolete data coexist in CR5M a revised mapping table is applied.

In the revised mapping table, an additional data area called mirroring address (MA) is appended to each entry. Its value tells the existence of mirroring data for current entry. If an entry does not have a mirroring data the value of MA area is NULL. Otherwise, an address will be stored in MA, which point to a mirroring table containing two areas. The first area is EPPN (expired physical page number), which records the physical page address for the obsolete data. The second area stores the physical address of mirroring data. It is called MPPN (mirroring physical page number). Fig. 4 shows an example of mapping table in CR5M. User data D_3' experienced an MW operation. The new data is stored in a page with physical address 42. MA area of D_3' is not NULL and points to the first entry of mirroring table. From the mirroring table, CR5M can find the old data and a duplication of D_3' at physical page address 40 and 160, respectively. After expired data reclaim the MA will be set to NULL and the corresponding entry in mirroring table will be deleted.

The CR5M increases performance and reliability in two aspects. First, the mean response time of a partial stripe update is reduced as additional reads and parity calculation are eliminated. Actually, these reads and parity computation are postponed to expired data reclaim process. The delay of reads can reduce the overall mean response time especially in non

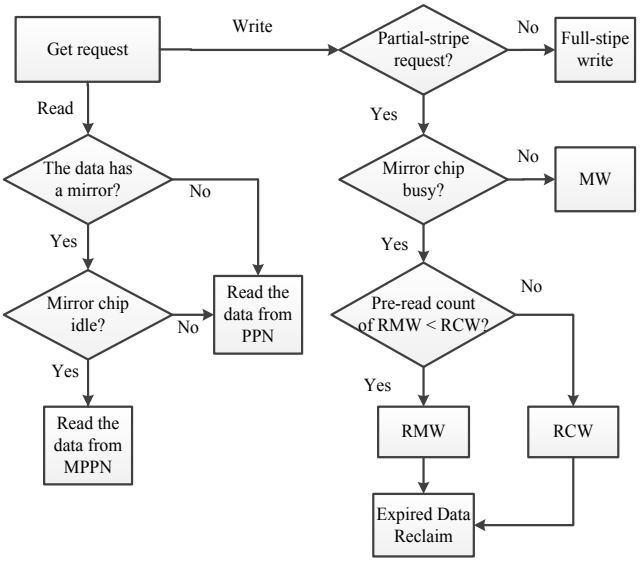

Fig. 5. The workflow of CR5M SSD.

data-intensive applications. Moreover, this delay is also helpful to reduce the number of parity calculation because only the latest version of frequently updated stripes is used for parity computation. Second, CR5M stores a duplication for partial stripe updates, leading to an improved read performance because request can be still served from the duplication if the original data is not available. Third, both the old data and updates are stored in CR5M. Hence, any one-channel data failures can be easily recovered. Furthermore, the duplication in mirroring chip implicitly increases the reliability of data in partial stripe updates before new parity is updated.

CR5M increases performance at the cost of one-chip capacity loss per channel. Assume that a CR5M consists of 4 channels with each having 5 chips. Storage efficiency per channel is 4/5. Considering an N-channel parity-encoded SSD, CR5M's storage efficiency would be *(N-1)/N*. Hence, a channel that has a larger number of chips would get a better storage efficiency.

Fig. 5 shows how write and read operations are carried out in CR5M SSD.

1) Write Policy

a) Full-stripe write will be applied if data size is bigger than one stripe. In case of full-stripe write no read operation will be performed and parity data is directly computed.

b) When request data size is smaller than one stripe a partial-stripe write will be carried out. In this scenario, CR5M will first check the status of mirroring chips on the channels where updates are going to be written. If the corresponding mirroring chips are idle, MW process is triggered. Otherwise, either RMW or RCW process will be proformed. For example, let us consider the case that D_6 in stripe 2 is going to be updated. The update of D_6 is D_6' (see Fig. 4). Assume that the mirroring chip on channel 0 is available. Hence, the MW process is applied. The D6' is first written onto chip 2 and mirroring chip (i.e., chip 4) concurrently (see Fig. 2). And then, its PPN 82 on chip 2 is recorded in the mapping table

and the address of corresponding mirroring table entry is set in the MA area. In mirroring table, the address of mirroring data is recorded in MPPN and the address of out-of-date D_6, 80, is recorded in EPPN. The new parity of stripe 2 will not be calculated in this procedure. Hence, the other pieces of data on this stripe can still be protected by D_6 and the old pairty. D_6' and its mirroring data are protected by each other.

c) In RMW or RCW scenario, the one whose read operation number is less will be adopted. As a new parity will be calculated an expired data reclaim operation will be invoked to reclaim the out-of-date data (i.e., data processed by MW operation) in this stripe. Fig. 4b illustrates the mapping table change after an expired data reclaim process for stripe 2. Assuming that D_7 is to be updated. And then D_6' and D_8 will be first read out to compute new parity for the stripe. Next, the new pairty P_2' and D_7' are written followed by updating corresponding entry in mapping table. In this stripe, D_6' was updated by MW. Thus, after the expired data reclaim process the MA area of D_6' is set to NULL and the associated entry in mirroring table is released. All the released pages are marked as invalided and they will be reclaimed in garbage collection.

d) Another scenario that the expired data reclaim operation will be invoked is that the available capacity in mirroing chip is lower than a predefined threshold. Typically, 2% of total capacity is overprovisioned for bad block replacement and garbage colletion [14]. In this research, the threshold is set as 2% of total chip capacity.

2) Read Policy

The mirroring chips also contribute to the read performance improvement because read requests can be sent to either the data chip or the mirroring chip if the piece of data resides on both chips. For instance, when a read request on D_3' arrives and D_3' has a copy on PPN 160 in chip 4, this read can be sent to chip 4 if chip 1 is busy or it can also be sent to chip 1 if chip 4 is not available.

IV. EXPERIMENTS

A. Experimental Setup

To evaluate the effectiveness of CR5M, an SSD simulator is implemented. Rather than developing a new simulator, our simulator is built based on a validated SSD simulator named SSDsim [6][17]. SSDsim is an event-driven, modularly structured, and highly accurate simulator for SSD. We extend the SSDSim by modifying several function modules including the trace file pre-processor, read request handler, write request handler, mapping table, allocation algorithm, and buffer management to support a handful of channel-level RAID architectures such as channel-RAID1 (CR1), channel-RAID4 (CR4), channel-RAID5 (CR5), and channel-level RAID5 with mirroring (CR5M). The simulator extracts requests from a trace file. Next, the requests are split to multiple sub-requests, which are distributed to multiple chips later. In the case of a write request, the simulator will generate several read/write sub-requests to update its corresponding parity. Only after all the sub-requests of a request are finished, the simulator marks the request as completed.

TABLE I. THE CHARACTERISTICS OF TRACES.

Trace Name	Write Ratio (%)	Ave.Size (KB)	Access Rate (req/sec.)	Duration (mins.)
Financial1	77.88	3.46	129	515
Radius9	88.46	6.8	57	35.2
ATTO	47.45	23.1	792.4	2.5
Build	45.71	6.5	372	15
Exchange	46.43	12.5	166	15

TABLE II. THE FIXED EXPERIMENT PARAMETERS.

Parameters	Values
Page read	20 μs
Page write	200 μs
Block erase	1.5 *ms*
Transfer one byte	25 *ns*
Page size	2 *KB*
Pages per block	64
Blocks per plane	2048
Planes per die	4
Dies per chip	4

TABLE III. THE VARIED EXPERIMENT PARAMETERS.

Conf.	Pure SSD	CR1	CR4 & CR5	CR5M
SSD1	*4cl-6cp*	*8cl-6cp*	*4cl-6cp*	*4cl-7cp*
SSD2	*6cl-4cp*	*12cl-4cp*	*6cl-4cp*	*6cl-5cp*
SSD3	*8cl-3cp*	*16cl-3cp*	*8cl-3cp*	*8cl-4cp*

Four real-world traces and a benchmark are selected to evaluate the performance of pure SDD, CR1, CR4, CR5, and the proposed CR5M SSD. The five traces and their characteristics are summarized in Table I. The selection of traces has been done so that different types of workloads are included. ATTO is a benchmark gathered from a PC with an NTFS file system by DiskMon [26][27]. ATTO generates the same amount of I/O requests irrespective of storage capacity and most of the random requests are the accesses to small number of sectors. The Build trace [28] was collected from the Microsoft Build Server production traces, which has roughly the same amount of read/write requests. The Exchange trace [28] was collected over a period a 24 hours at Microsoft Exchange Server using the event tracing for Windows framework. The access rate of Exchange is much smaller than that of Build. Financial1 [29] is an I/O trace from OLTP application running at a financial institution. The Radius trace [28] was collected for Radius authentication server. The Financial1 and Radius are write-dominant traces.

Table II illustrates the flash configuration used in experiments, whereas Table III shows the different number of channels and chips used for simulation. Three different sets of configurations, *SSD1*, *SSD2*, and *SSD3* are studied. In these three sets of configurations the number of channels varies from

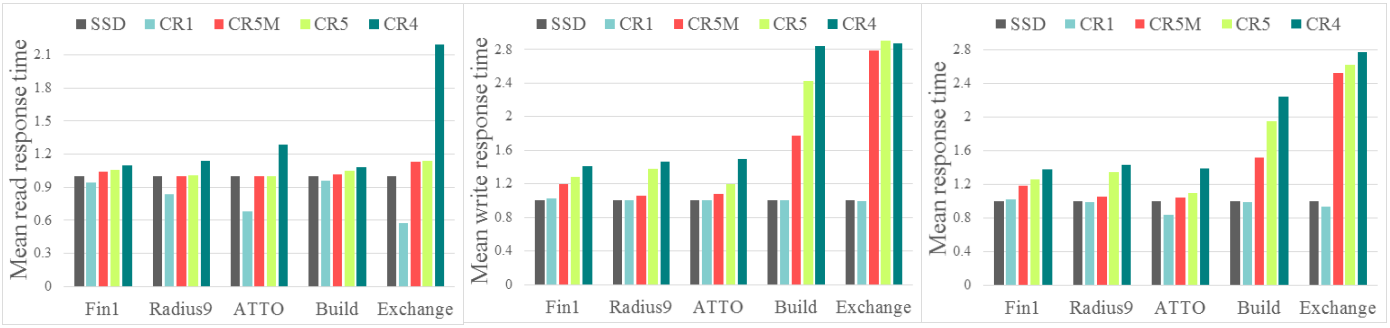

Fig. 6. Performance comparisons on SSD1.

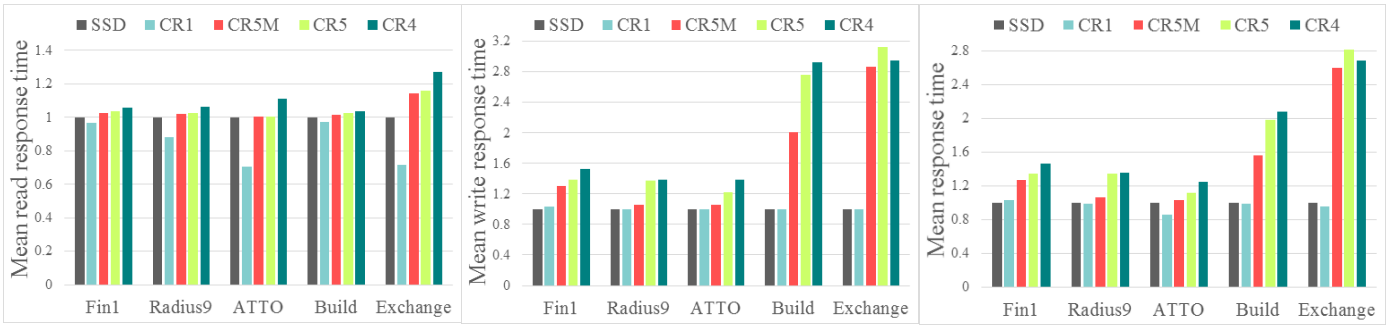

Fig. 7. Performance comparisons on SSD2.

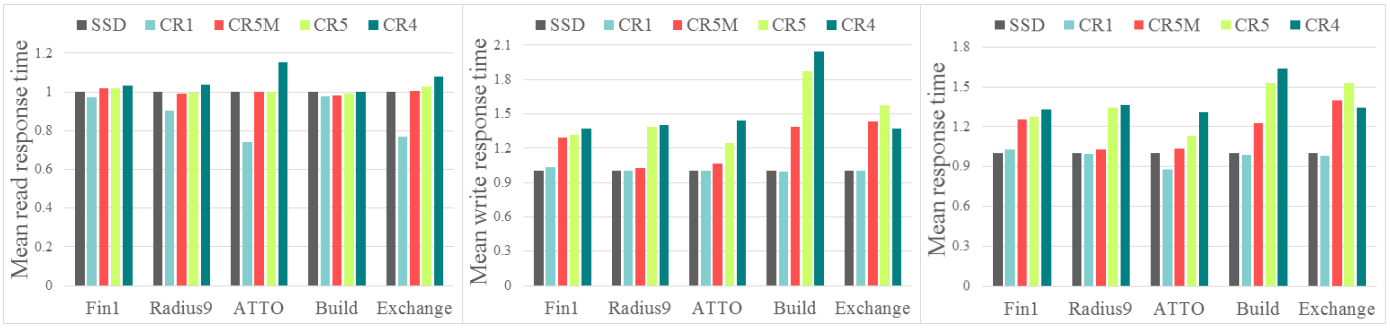

Fig. 8. Performance comparisons on SSD3.

4 to 8 so that a comprehensive understanding of its impact can be gotten. Due to the mirroring requirement the number of channels in CR1 is as twice as that in other architectures. In all the experiments, the total usable capacity keeps the same. In table III, while '*cl*' means the channel number in an SSD, '*cp*' stands for the chip number on each channel. For example, in *SSD1* CR5M configuration, 4cl-*7cp* means an SSD has 4 channels with each containing 7 chips. In all the experiments, 5% of flash memory capacity is preserved as overprovisioned space for bad block management and garbage collection.

B. Real-World Workloads

Fig. 6, 7 and 8 show the overall performance in terms of mean response time in *SSD1*, *SSD2*, and *SSD3* configurations. All the values are normalized to that in the pure SSD configuration. It is clear that the mean response time of read operations does not change too much in CR5 and CR5M configurations compared with that in the pure SSD. For CR1 configuration, a noticeable improvement of mean response time can be found in almost every trace. The reason behind this is that the mirroring chips in CR1 increase the data availability so that the throughput of read is enlarged. Moreover, CR4 experience the worst read performance among all configurations, especially when the number of channel is small. Compared with CR5, the read performance of CR5M increases at most 3%. Intuitively, CR5M should have a big improvement in read performance due to the mirroring chip on each channel. However, in real situations the read sub-requests that are distributed to the mirroring chip is only 1.2% of total requests. Hence, the read performance does not gain too much in the mirroring chip.

For write performance, the pure SSD and CR1 configurations outperform the other three configurations (i.e., CR4, CR5, and CR5M) under all the traces. This is due to the fact that CR4, CR5, and CR5M have an extra overhead of parity calculation and updating. CR4 exhibits the worst write performance as it uses a dedicated parity channel. The parity channel becomes its performance bottleneck because each stripe write or update results in a parity channel write. Under the Build and Radius9 traces, CR5M improves 27.2% and 25.6% mean write response time compared with CR5, respectively. This improvement gains from the MW of CR5M, which takes a less time to serve partial-stripe requests than

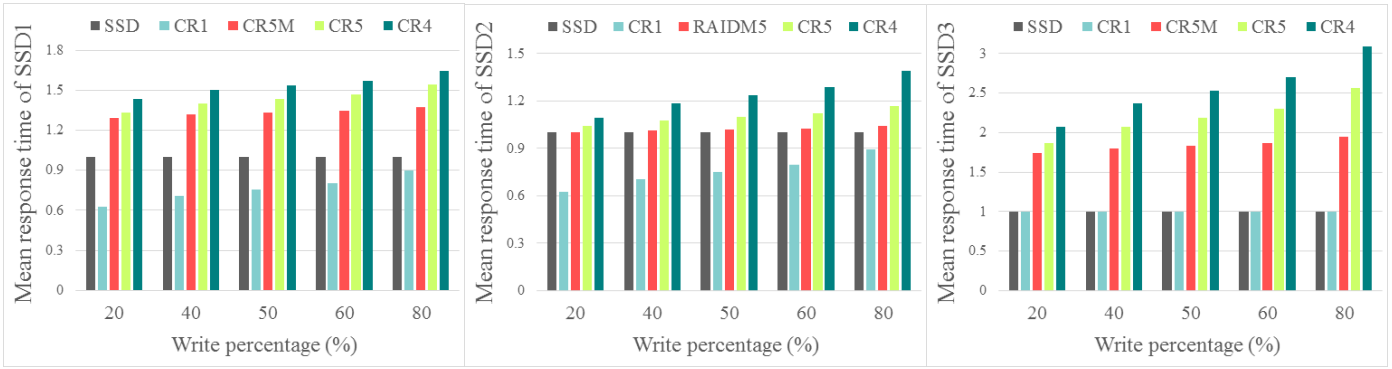

Fig. 9. Impact of write percentage.

Fig. 10. Impact of average request size.

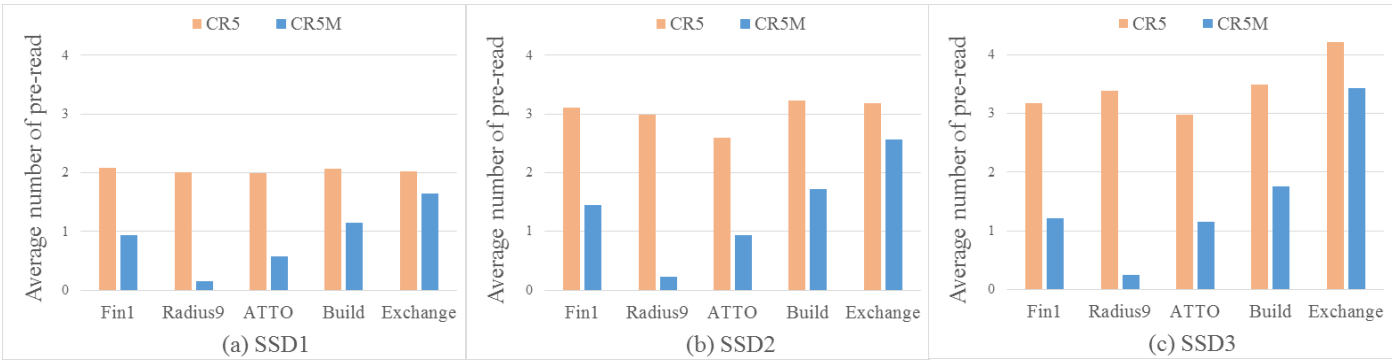

Fig. 11. Overhead of pre-read operations.

RMW or RCW. Similar to the simulation results of write performance, the overall mean response time of CR5M is better than CR4 and CR5. For example, under Financial1, Radius9, ATTO, Build and Exchange trace, CR5M improves 6%, 25.8%, 8.5%, 22.6% and 8.6% compared to CR5. Besides, CR1 achieves the best performance, which profits from the doubled flash memory capacity. CR4, on the contrary, performs the worst because of the dedicated parity channels.

C. *Synthetic Workloads*

A set of synthetic workloads is also used to evaluate the CR5M architecture. In particular, we evaluate the impact of write percentage and average request size on overall performance. All the experimental results are normalized to that of a pure SSD.

Fig.9 shows the impact of write request percentage on performance. The default average request size is set to 16KB and the access rate is configured to 240 requests per second. We vary the write percentage from 20% to 80%. CR5M exhibits the best performance in the 80% write scenario. It outperforms CR5 by up to 24.1%. As MW can boost write performance a lot, CR5M provides a large improvement in high write ratio scenario.

Fig. 10 presents the impact of average request size on performance. The default write-ratio is set to be 60% and the default access rate is configured to 240 requests per second. Compared with CR5, the smaller the average request size, the more improvement can be obtained by CR5M. In the best situation, CR5M can gain improvement by 31.7%. Clearly, when request size increases, the gap between CR5 and CR5M becomes small.

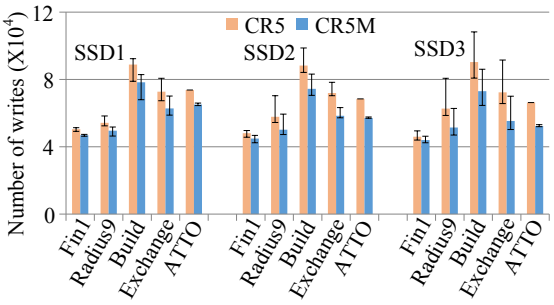

Fig. 12. Average number of writes per channel.

D. *Parity Pre-Read Overhead*

For partial-stripe write/update, a number of pre-read operations have to be performed so that new parity can be calculated. The number of pre-read operations should be as less as possible so that the overhead of parity calculation is minimized. Fig. 11 shows the average number of pre-read operations for each write request. The pre-read operations are the extra operations for parity updating. Unlike RMW and RCW, MW does not generate any pre-read operation so that the average number of pre-read operations in CR5M is reduced due to the MW. It is clear that under all the traces, the number of pre-read operations in CR5M is smaller than that of CR5. On average CR5M reduces the number of pre-reads by 56%. In Radius9 trace CR5M reduces the number significantly. The reason behind this is that Radius possesses a lower access rate, which results in a larger number of MW.

E. *Wear-Leveling Evaluation*

The number of writes a chip received is a good indicator of wear-leveling as a large number of writes/updates can lead to a increased number of P/E cycles. Fig. 12 shows the average number of writes per channel when CR5 and CR5M are used, respectively. The height of a bar represents the average number of writes among all channels. The upper and lower cap of an error bar shows the largest and the smallest number of writes that a channel receives in a single SSD. Hence, the difference between the upper and lower cap gives the largest difference of wear out degree among channels. It is obvious that the number of writes a channel receives in CR5M is smaller than that in CR5 under all the real-world traces. On average, CR5M can reduce the number of writes per channel by 14% compared with CR5. The reason behind this is that the MW provided by mirroring chips absorbs lots of parity updates in partial-stripe updates. Especially, for stripe who has a high updating frequency, only the parity of the last update (i.e., before expired data reclaim operation) will be calculated and write onto flash.

V. CONCLUSIONS

Due to aggressive scaling down technology flash memory reliability continuously decreases, which threatens data integrity and reliability in SSDs [5]. A typical approach to addressing this problem is to apply an ECC scheme, which can combat bit errors so that data integrity can be guaranteed [3][5]. However, each ECC scheme has its own capacity limitation, above which it can no longer work. [5]. Thus, a data redundancy mechanism is greatly needed to protect data under permanent errors. RAID structures have been successfully applied in disk arrays to improve data reliability and integrity for server-class applications [9]. However, there are many cases [21][23] where only one SSD can be deployed due to space and energy constraints. Yet, data reliability in such applications like wireless healthcare systems [21] and mobile military applications [23] is critical. Fortunately, the internal hierarchical architecture of an SSD provides us with an opportunity to employ a RAID-like structure at the channel level. However, directly applying a RAID format at the channel level faces several challenges. Firstly, frequent parity updates largely accelerate aging of an SSD whose P/E cycles are already limited [3]. Secondly, the parity updates increase an SSD's write amplification, which leads to a poorer SSD performance. In this paper, to fully understand the impact of RAID architecture on a single SSD's performance and reliability, we first implement and study several channel-RAID organizations. We found that CR1 consistently outperforms the other channel-RAID architectures. For example, in SSD1 configuration compared to a non-RAID SSD CR1 can improve the I/O performance by up to 19%. Obviously, it improves performance at the cost of 50% capacity loss. CR4 and CR5 use parity to enhance data reliability. In SSD1 configuration, CR5 outperforms CR4 in the range of 9% ~ 26%. The reason behind this is that CR4 adopts a dedicated parity channel, whereas CR5 distributes parities on all the channels. The dedicated parity channel in CR4 becomes a performance bottleneck as each data update results in a parity update on the parity channel. Next, a mirroring-powered channel-RAID5 structure called CR5M is proposed. It can be applied to a single SSD. CR5M introduces a mirroring chip on each channel to accelerate small writes/updates by reducing the overhead of frequent parity updating. Moreover, the mirroring chips provide a data protection for the small writes/updates, which are not covered by present parity. Comprehensive experimental results show that compared with CR5, CR5M achieves a performance gain up to 25.8%. Besides, CR5M reduces the average writes per channel in the range of 4.5% ~ 23.6% compared with CR5.

In the future, we will implement and study the channel-RAID architecture on a hardware evaluation board where real flash chips are employed. The increased overhead caused by channel-RAID data management and energy consumption will be comprehensively studied.

VI. ACKNOWLEDGEMENT

This work is sponsored in part by the U.S. National Science Foundation under grant CNS-(CAREER)-0845105 and Key Technologies R&D Program of Anhui Province (China)-11010202190.

References

[1] N. Agrawal, V. Prabhakaran,T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, "Design Tradeoffs for SSD Performance," in Proc. USENIX Annual Technical Conf., pp. 57-70, 2008.

[2] SZ. Chen, and D. Towsley, "The design and evaluation of RAID 5 and parity striping disk array architectures," Journal of Paralel and Distributed Computing, Vol. 17, No. 1, 1993

[3] J. Cooke, "The inconvenient truths about nand flash memory," Micron MEMCON, June, 2007

[4] Y. Cai, F. H. Erich, M. Onur, and M. Ken, “Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis,” In IEEE DATE, 2012.

[5] E. Deal, “Trends of nand flash memory error correction,” Cyclic Design, June, 2009

[6] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang. "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity." in Proc. Int’l Conf. Supercomputing (ICS), pp. 96-107, 2011

[7] L.M. Grupp, D. John, and S. Swanson, “The harey tortoise: managing heterogeneous write performance in SSDs,” In USENIX ATC, 2013

[8] L.M. Grupp, A.M. Caulfield, J. Coburn, S. Swanson, “Characterizing flash memory: anomalies, observations, and applications,” In IEEE/ACM MICRO, 2009

[9] R. Bilton, “Good News for Consumers: Solid State Drives prices are dropping,” http://www.zdnet.com/blog/storage/good-news-for-consumers-solid-state-drive-prices-are-dropping/1706, June 2012.

[10] S. Im, D. Shin, “Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD,” IEEE Transactions on Computers, Vol. 60, Issue1, pp. 80-92, Jan. 2011.

[11] A. Kadav, M. Kalarishnan, V. Prabhakaran, and D. Malkhi, “Differential RAID: Rethinking RAID for SSD Reliability,” ACM Transactions on Storage, Vol. 6, Issue 2, July 2010.

[12] Y. Lee, S. Jung, and Y.H. Song, “FRA: A Flash-aware Redundant Array of Flash Storage Devices,” in Proc. 7th IEEE/ACM Int’l Conf. on Hardware/Software Codesign and System Synthesis, pp. 163-172, 2009.

[13] B. Mao, H. Jiang, D. Feng, S. Wu, J. Chen, L. Zheng, and L. Tian, “HPDA: A Hybrid Parity-based Disk Array for Enhanced Performance and Reliability,” in Proc. IEEE Int’l Symp. on Parallel & Distributed Processing (IPDPS), pp. 1-12, 2010.

[14] Micron. (2010, Oct.) Bad block management in NAND flash memory. [Online]. Available: http://www.micron.com/products/support/technicalnotes/

[15] Y. Du, F. Liu, Z. Chen, and X. Ma, “Wele-RAID: a SSD- based RAID for System Endurance and Performance,” in Proc. 8th IFIP international conference on Network and parallel computing (NPC), pp. 248-262, 2011.

[16] J. Hui, X. Ge, X. Huang, Y. Liu, and Q. Ran, “E-HASH: An Energy-Efficient Hybrid Storage System Composed of One SSD and Multiple HDDs,” in Proc. Springer-Verlag Berlin Heidelberg, pp. 527-534, 2012.

[17] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and C. Ren, “Exploring and Exploiting the Multi-level Parallelism Inside SSDs for Improved Performance and Endurance,” IEEE Transaction on Computers, Vol. 62, No. 6, pp. 1141-1155, June 2013.

[18] F. Chen, R. Lee, and X. Zhang, “Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing,” in Proc. High Performance Computer Architecture (HPCA), pp. 266-277, February 2011.

[19] S. Park, S. Ha, K. Bang and E. Chung, “Design and analysis of flash translation layers for multi-channel NAND flash-based storage devices,” IEEE Transactions on Consumer Electronics, Vol. 55, Issue3, pp. 1392-1400, Aug. 2009.

[20] A.R. Abdurrab, T. Xie, and W. Wang, “DLOOP: A Flash Translation Layer Exploiting Plane-Level Parallelism,” In IEEE IPDPS, 2013.

[21] S. Im and D. Shin, “Delayed Partial Parity Scheme for Reliable and High-Performance Flash Memory SSD,” Proc. IEEE Symposium on Mass Storage Systems and Technologies (MSST), May 3-7, 2010.

[22] D.A. Patterson, G. Gibson, and R.H. Katz, “A Case for Redundant Arrays of Inexpensive Disks (RAID),” ACM SIGMOD 88, Vol. 17, No. 3, June, 1988

[23] J.M. Smith, “The doctor will see you ALWAYS,” IEEE Spectrum, Vol. 48, No. 10, pp. 56-62, October 2011

[24] G. Stefano, A. Cabrini, O. Khouri, and G. torelli, “On-Chip Error Correcting Techniques for New-Generation Flash Memories,” Proceedings of the IEEE, Vol 91, pp. 602-616, 2003

[25] Department of Defense Mobile Device Strategy, http://www.defense.gov/news/dodmobilitystrategy.pdf, 2012

[26] ATTO, http://www.attotech.com/disk-benchmark/.

[27] DiskMon for Windows v2.01, http://technet.microsoft.com/en-us/Sysinternals/Bb896646.aspx

[28] SPC, Storage Performance Council I/O Repository, http://iotta.snia.org/traces/.

[29] K. Bates, and B.McNutt, “Financial1 and Financial2 Traces,” http://traces.cs.umass.edu/index.php/Storage/Storage