Enhancing SSD Reliability through Efficient RAID Support

Jaeho Kim[1], Jongmin Lee[1], Jongmoo Choi[2], Donghee Lee[1], and Sam H. Noh[3]

[1]University of Seoul, {kjhnet10, jmlee, dhl_express}@uos.ac.kr
[2]Dankook University, choijm@dankook.ac.kr
[3]Hongik University, http://next.hongik.ac.kr

Abstract

A serious problem with current SSDs is its low reliability due to their primary component, flash-memory, that has high error rate and limited erase count. Adopting RAID architecture is a reasonable way to increase reliability of SSDs. In this paper, we propose Dynamic and Variable Size Striping-RAID (DVS-RAID) that dynamically constructs a variable size stripe based on arrival order of write requests such that write requests are sequentially written to a stripe improving the performance and lifetime of SSDs. To increase the reliability of small writes without making use of non-volatile RAM, DVS-RAID employs variable size striping, which constructs a new stripe with data written to portions of a full stripe and writes a parity for that partial stripe. We implement DVS-RAID in the DiskSim SSD extension, and experimental results based on trace-driven simulations show that DVS-RAID outperforms the conventional RAID-5 scheme in terms of performance and lifetime of SSDs.

1 Introduction

This paper considers using RAID to increase the reliability of flash memory based SSDs. New developments in flash memory, in particular, multi-level cell (MLC) and triple-level cell (TLC) flash memory, have brought about higher density. However, with it the program/erase (P/E) cycles for these technologies have come down rapidly, from 100,000 for single-level cell (SLC) flash memory down to 10,000 and 2,500 P/E cycles for MLC and TLC flash memory, respectively [6]. This in turn exacerbates the issue of reliability in flash based products as the bit error rate (BER) is strongly correlated to the wear down of P/E cycles [4, 6].

In order to improve reliability, conventional SSDs record error correction code (ECC) in out-of-band (OOB) flash memory. However, ECC has some intrinsic limitations. First, its detection and correction level is highly affected by the size of the ECC [3]. Larger ECCs allows higher level of data recovery but requires larger OOB area, where size is quite limited. If the number of bits in error is beyond the ECC capacity, then there is nothing that can be done, and this is becoming a reality [9]. In similar context, page-, block-, or chip-level errors cannot be corrected. Hence, RAID based corrective measures have been proposed [5, 7, 8]. However, these approaches each have their own limitations as we discuss later.

In this paper, we propose a technique for providing RAID-5 level or better reliability for flash based SSDs by constructing RAID at the chip level. The technique, which we call Dynamic and Variable Size Striping-RAID (DVS-RAID), does away with the inherent small write problem associated with conventional RAID-5 by dynamically constructing a variable size stripe based on arrival order of write requests. Though the idea of dynamic striping has

APSys '12, July 23-24, 2012, Seoul, S. Korea

been known for disk-based RAID system, we evaluate how much performance enhancement can be achieved and how much overhead is required for applying RAID configuration to SSDs. Furthermore, DVS-RAID improves the reliability of SSDs through variable size striping without any hardware support. While conventional RAID methods adapted to flash memory must write stripes in whole to provide RAID-5 reliability, the variable size striping scheme allows for small writes to construct a partial stripe and to write the parity for this partial stripe. Consequently, there can be multiple partial stripes in a full stripe. This is especially important as a physical stripe in today's SSDs may be composed of a large number of chips and thus a substantial number of write requests may not be able to fill the whole stripe by itself or within a safe time period. Through experiments with the DiskSim SSD extension, we explore the benefits and overhead of DVS-RAID compared to the conventional RAID-5 scheme.

The rest of the paper is organized as follows. In the next section, we discuss how RAID is supported with SSDs. We also briefly discuss works related to this study. In Section 3, we describe the proposed DVS-RAID in detail. Then, in Section 4, we evaluate DVS-RAID via trace-driven simulations using the DiskSim SSD extension [10]. We show that DVS-RAID improves performance of SSDs by 24% and its life-span by 28% for realistic workloads compared to the conventional RAID-5 approach. Finally, we conclude with Section 5.

2 SSD and RAID

Let us start by reviewing the components that comprise an SSD with RAID support using Fig. 1(a), which shows the internal structure of an SSD and how the data would be dispersed among the chips within the SSD. In a typical SSD, there are N_C flash memory chips, and in each chip there are multiple blocks each associated with a Physical Block Number (PBN). In each block, there are multiple pages, each associated with a Physical Page Number (PPN). A stripe consists of N_C pages, each of which belongs to each individual chip. Typical SSDs today employ a large N_C value typically 10∼16 chips, and more in some SSDs, though in Fig. 1 we use

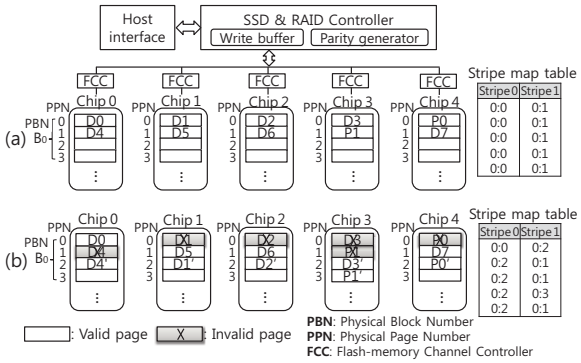


Figure 1: (a) Internal structure of conventional RAID-5 supporting SSDs, with initial data allocation and (b) how data and parity would be allocated as updates occur.

an example with only 5 chips. Dx and Px, in the figure, denote user data and parity, respectively. We emphasize that the focus of this study is in presenting a management scheme used at the SSD & RAID controller component in Fig. 1(a).

RAID, in current SSDs, are supported as follows [5, 8]. Take the initial situation in Fig. 1(a) where D0∼D7 are user data and there is a parity per stripe. Stripe 0 consists of pages D0∼D3 and P0, while Stripe 1 consists of pages D4∼D7 and P1. The Stripe map table in the controller holds information regarding each stripe where the number pairs represent the PBN and the PPN within the chip.

Assume data pages D1 through D4 are updated. Note that unlike disk-based RAID-5 where each old strip would be overwritten, this cannot happen with flash memory based SSDs. Instead of overwriting, new data must be written to a new location, and to employ every chip as a RAID component, the new data must be written on the same chip of the original data. Then using either read-modify-write (as for Stripe 1) or reconstruct-write (as for Stripe 0), existing data must be read to calculate the new parity, so that the new data and parity can be written to the chips. The Stripe map table is then updated to reflect the changes as shown in Fig. 1(b).

There are limitations to this approach. First, whether read-modify-write or reconstruct-write is employed, reading of existing data must precede new parity calculations. This is also true for traditional disk-based RAID-5 systems. Second, once

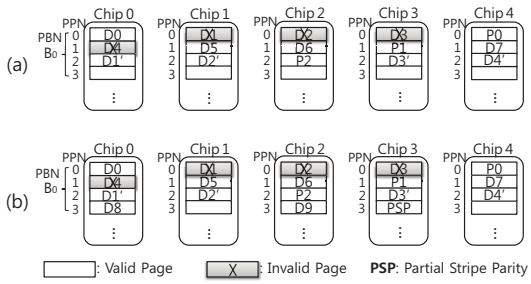


Figure 2: Write sequence for DVS-RAID as (a) data pages D1-D4 are modified and as (b) new partial stripe data D8 and D9 are written.

a data and parity page is allocated to a particular chip, this relation is fixed. Hence, if a particular page is written with higher frequency, then that particular chip will be written to more frequently. Also, the chip in which the parity page resides is more prone to wear out as it must be written to more frequently. These fixed relations eventually lead to higher cleaning costs and decreased lifetime of the SSD. Third, when non-existing data is written, the data cannot be written until the stripe becomes full, leaving open a window of vulnerability. For example, if new data D8 and D9 arrive, in our example above, a parity page cannot be calculated for these pages that form a partial stripe, and thus these pages cannot be written until a full stripe is formed, that is, another two new pages arrive.

Methods such as Partial Parity Cache (PPC) [5], Flash-aware Redundancy Array (FRA) [8] and the Lifespan-aware scheme [7] have been proposed to resolve these limitations. To reduce the cost of parity update of RAID-5, FRA avoids writing parities at time critical points postponing them to idle periods by making use of dual mapping tables. In so doing, FRA reduces write response time compared to RAID-5 [8]. However, failures at partial stripe write points cannot be recovered as parity calculations are delayed until a full stripe is written. PPC is an architecture that is similar to RAID-5, but resolves partial writes by generating partial parity for partial stripes. However, this is done by making use of non-volatile RAM (NVRAM), which is an added hardware component. The Lifespan-aware scheme is another proposal to enhance the reliability of SSDs through RAID support. As flash memory BER changes with more wear, the number of parity pages is adjusted to accommodate the error rate of the current SSD, increasing with longer usage. This scheme also requires that the parity be cached in NVRAM [7]. The fact that parity must be stored in costly NVRAM is a significant drawback of the last two schemes.

3 Dynamic and Variable Size Striping-RAID

In this section, we present a technique, which we call Dynamic and Variable Size Striping-RAID, that supports RAID-5 reliability for all data sizes, that is, full or partial stripes. Our scheme does not require any additional hardware component.

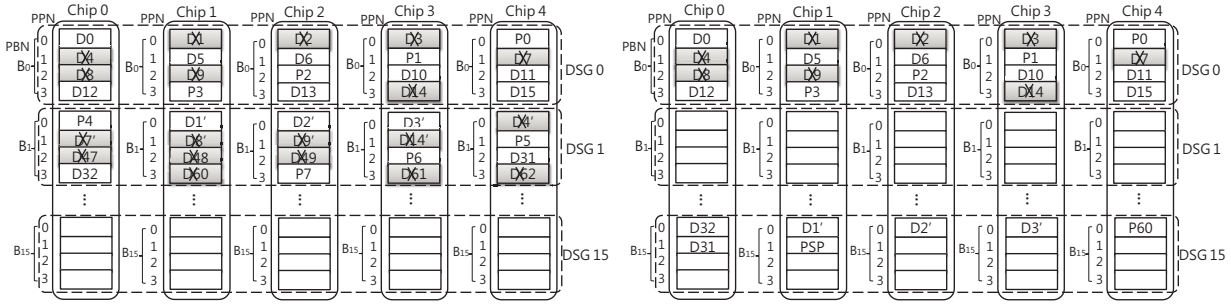
3.1 DVS-RAID and Parity Overhead

Dynamic and Variable Size Striping-RAID (DVS-RAID), the scheme that we propose, has two features. First, every strip that comprises a stripe always has the same PPN number. A stripe is constructed based on arrival order of write request regardless of LBN. Second, DVS-RAID employs a variable size striping scheme that can construct a partial stripe with data of a small write request, which we define to be a write request smaller than the full stripe size minus 1 (the minus 1 for the parity). As a result, multiple partial stripes can exist in a full stripe.

Let us go through an example starting from the same Fig. 1(a) with valid user data, D0-D7. With DVS-RAID, all pages of PPN 0 and 1 comprise Stripe 0 and 1, respectively. As D1$\sim$D4 are modified, the controller simply calculates the new parity for these pages, writes them on the pages with PPN 2 along with the parity value as shown in Fig. 2(a). After writing, the controller simply marks the old pages as obsolete. There is no need to read the old pages.

Furthermore, all chips are written to evenly even if particular pages are more frequently written to, including the parity page. In case of RAID-5, an entry of the Stripe map table consists of a PBN and PPN pair. Note that the chip number is not needed as the LBN of the data designates the chip number. However, for DVS-RAID, since a data page may be allocated to any chip, the Stripe map table must comprise the Chip ID, PBN, and PPN.

Now consider the case when only part of the stripe is written. Continuing with the previous ex-



(a) Before cleaning (b) After cleaning

Figure 3: SSD internals with DVS-RAID (a) before cleaning and (b) after cleaning.

ample, assume again that D8 and D9 are being newly written. Then DVS-RAID does not wait for data of the entire stripe to arrive, but constructs a variable size stripe with the written data by generating a partial parity. Then, the data and parity of the stripe are written right away gaining RAID-5 reliability. In our example, shown in Fig. 2(b), D8 and D9, respectively, is allocated to PPN 3 of Chips 0 and 2, while the parity for these data, denoted PSP in the figure, is written to Chip 3. Note that we skip Chip 1 as for this particular PPN, Chip 1 is the position of the full stripe parity. In our scheme, the page in this position is either used to store the full stripe parity, as it normally should, or it is used to store the last partial parity in that PPN. This will allow for the parities to be evenly distributed as much as possible. As more data arrive, say D10, the last parity is calculated with it and written to the skipped Chip 1. This method does waste some space for storing multiple parities for partial stripes, but these are reclaimed during garbage collection, hence there is no permanent loss of capacity. However, as multiple parities take up space, garbage collection may have to be performed more frequently, increasing the number of write requests. We will later quantitatively evaluate how many extra writes are incurred by employing variable size striping.

This variable size striping scheme increases reliability in two aspects. First, it protects data of small write requests immediately after they and their associated parities are written. In contrast, the RAID-5 scheme can not protect its data until the data portion of the full stripe is filled, its parity calculated, and the full stripe is written. Second, two or more errors can be recovered if they fall into different partial stripes that are protected by different parities.

Table 1: Parameter of SSD simulator

Parameter	Value	Parameter	Value
Page size	4KB	Page write	200us
Block size	256KB	Block erase	1.5ms
Page read	25us	Page Xfer latency	100us

Table 2: Characteristics of I/O workload

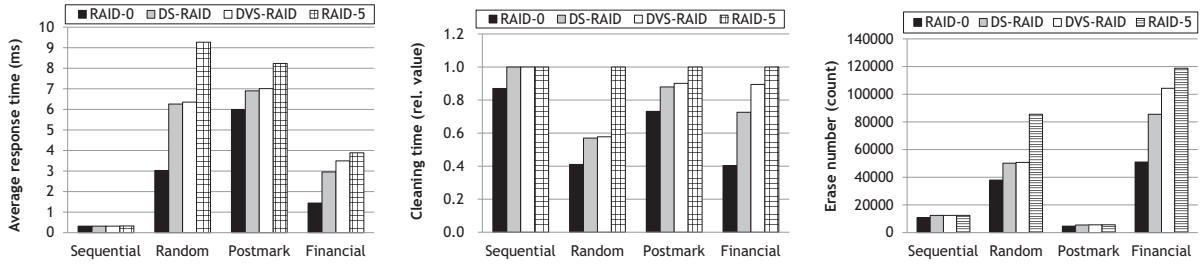
Workload	Avg. Wrt Req. Size	Aggr. Wrt Req
Sequential	9.1KB	18GB
Random	12.6KB	18GB
Postmark	1.6MB	11.1GB
Financial	10.9KB	28.8GB

This results in DVS-RAID providing stronger reliability than RAID-5.

3.2 Cleaning operation of DVS-RAID

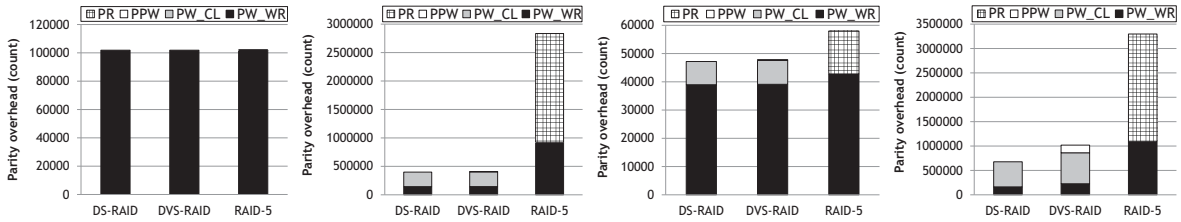
Since flash memory suffers from out-of-place-update, updated data must be written to free pages. If there is no free space to write to, a cleaning operation is invoked to make free space. Fig. 3 depicts the situation within the SSD before and after cleaning in DVS-RAID. To describe the cleaning process, we define a notion of a Dynamic Stripe Group (DSG). A DSG is composed of physical blocks that comprise a stripe, which means that these physical blocks are all of the same block number. For example, as shown in Fig. 3, block 0 of all chips comprise DSG0, block 1 of all chips comprise DSG1, and so on.

The cleaning process of DVS-RAID proceeds as follows. First, a DSG with the smallest number of valid pages is chosen as the victim DSG. In our example in Fig. 3, DSG1 is selected. Note that for cleaning purposes, at least one empty DSG must always be available. In our example, this is DSG15.



(a) Average Response time (b) Cleaning time (c) Erase count

Figure 4: Performance results



(a) Sequential (b) Random (c) Postmark (d) Financial

Figure 5: Analysis of parity overhead (PR, PPW, PW_CL, and PW_WR denotes page reads, partial stripe parity writes, parity writes during cleaning, and parity writes for write requests, respectively.)

The next step is to copy the valid pages from the selected DSG to the empty DSG. Note here that only the data pages are copied, and that the parity pages are not. The old parity pages are simply discarded, and new parity pages are calculated with only the valid pages. In our example, there are 5 valid pages, and among them D32, D1', D2', and D3' form a stripe and with them, a new parity page, P60, is calculated and stored in Chip 4. The last remaining valid page, D31, forms a partial stripe and a PSP is calculated for this page. D31 and its corresponding PSP is stored in Chips 0 and 1, respectively as shown in Fig. 3(b).

4 Evaluation

For evaluation, we implemented the DVS-RAID scheme by modifying the Disksim SSD extension [10]. Table 1 presents the simulation parameters where 5% of the SSD is utilized as over-provisioned space for cleaning operations. We assume that 8 flash memory chips are configured for RAID (i.e., 8-channels), implying that a full stripe consists of 8 pages (32KB). We compare DVS-RAID with RAID-0 and RAID-5 in terms of I/O response time, cleaning time and parity management overhead. RAID-0, which is typically used in conventional SSDs, does not maintain parity and only stripes data to exploit the parallelism within SSDs. For comparison, we also implemented DS-RAID (Dynamic Striping) that employs dynamic striping but not variable size striping. Four workloads are used as summarized in Table 2. The first two workloads are synthetic ones that access data sequentially one by one or randomly with uniform distribution. The Postmark workload is a file system benchmark with significant sequential I/Os and an average write request size of 1.6MB [2]. Finally, the Financial workload is a random write intensive workload from OLTP applications running at financial institutions [1].

Fig. 4(a) shows the average response time results. The *x*-axis denotes all the evaluated schemes per workload, while the *y*-axis represents the average response time. For DVS-RAID, a new partial stripe is constructed with data of the small write request when the inter-arrival distance between two requests exceeds 50 milliseconds. As observed from the figure, RAID-0, which does not handle parity, offers the best performance. For the sequential workload, RAID-5 does not show much performance degradation as most writes are handled through full striping. However, it performs poorly for the random workload due to parity management overhead for random updates. In contrast, both DS-RAID and DVS-RAID reduce this overhead as random writes

are converted to sequential ones as many of the individual writes are processed simply in the order they arrive. Similar results are also observed for the Postmark and Financial workloads where DVS-RAID performance improves by roughly 16% and 24%, respectively, compared to RAID-5. Note that DS-RAID supports reliability comparable to RAID-5 while RAID-0 does not, and that DVS-RAID further enhances reliability by writing parities for small writes.

Comparisons of cleaning times and erase counts are depicted in Fig. 4(b) and 4(c). The overall observations are similar to those of Fig. 4(a): 1) RAID-0 performs best as it does not carry out parity management, 2) RAID-5 performs worst, especially for the random workload, as small writes incur more parity writes, and this in turn, results in more frequent cleaning, and 3) DS-RAID reduces cleaning overhead considerably, while DVS-RAID sacrifices some cleaning time for reliability. Erase counts show similar trends to cleaning times except that the sequential and Postmark workloads, whose request size is generally large, have much smaller erase counts than the other two, whose write requests are small and random.

Fig. 5 distinguishes the various components involved in managing the parity. In particular, for RAID-5, the parity overhead consists of page reads needed for parity calculations (denoted PR) and the parity writes for write requests (denoted PW_WR). In contrast, for DS-RAID and DVS-RAID, there is no need to read pages during parity management. However, parity writes are invoked during the cleaning process, which we denote as PW_CL. DVS-RAID has an additional component in that multiple parities may be written and we denote this component as PPW (Partial Parity Writes). However, in the figure comparing DS-RAID and DVS-RAID, we can see that extra writes incurred by variable size striping is negligible except for the Financial trace that has many small random writes.

5 Conclusion

In this work, we proposed a novel flash-aware RAID scheme, called DVS-RAID, that dynamically forms a variable size stripe based on the arrival order of write requests by exploiting the out-of-place-update characteristic of flash memory, the basic component of SSDs. Experimental results showed that DVS-RAID can effectively overcome the small write problem, greatly improving the parity management overhead of RAID-5. There are still many other issues that need to be looked into more closely, namely, efficient failure-recovery algorithms, effectiveness of the algorithm with real SSD error behavior, and RAID-conscious wear leveling and victim block selection policies. We are currently looking into these issues.

Acknowledgments

We would like to thank the anonymous reviewers and colleagues for their constructive comments. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. R0A-2007-000-20071-0), by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No. 2009-0085883), and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0025282).

References

[1] UMASS TRACE REPOSITORY. *http://traces.cs.umass.edu*.

[2] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *2008 USENIX ATC*.

[3] E. Deal. Trends in NAND flash memory error correction. Cyclic Design, White Paper, Jun. 2009, http://www.cyclicdesign.com/whitepapers/Cyclic_Design_NAND_ECC.pdf.

[4] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf. Characterizing flash memory: anomalies, observations, and applications. In *Proceedings of MICRO 42*.

[5] S. Im and D. Shin. Flash-Aware RAID Techniques for Dependable and High-Performance Flash Memory SSD. *IEEE Transactions on Computers*, 60(1):80 –92, Jan. 2011.

[6] M. G. Laura, D. D. John, and S. Steven. The Bleak Future of NAND Flash Memory. In *Proceedings of FAST '12*, 2012.

[7] S. Lee, B. Lee, K. Koh, and H. Bahn. A lifespan-aware reliability scheme for RAID-based flash storage. In *Proceedings of the 2011 ACM SAC*.

[8] Y. Lee, S. Jung, and Y. H. Song. FRA: a flash-aware redundancy array of flash storage devices. In *Proceedings of CODES+ISSS '09*, 2009.

[9] M. Mariano. Ecc options for improving nand flash memory reliability. Micron, 2012, http://www.micron.com/support/software/~media/Documents/Products/Software%20Article/SWNL_implementing_ecc.ashx.

[10] V. Prabhakaran and T. Wobber. SSD Extension for DiskSim Simulation Environment. *http://research.microsoft.com/en-us/downloads/b41019e2-1d2b-44d8-b512-ba35ab814cd4*.