Parity-Stream Separation and SLC/MLC Convertible Programming for Lifespan and Performance Improvement of SSD RAIDs

Yoohyuk Lim
College of Software
Sungkyunkwan University
Suwon, South Korea

Jaemin Lee Cassiano Campes Euiseong Seo
College of Information and Communication Engineering
Sungkyunkwan University
Suwon, South Korea

Abstract

To reduce the performance and lifespan loss caused by the partial-stripe writes in SSD RAIDs, we propose two schemes: parity-stream separation and SLC/MLC convertible programming. Parity-stream separation splits the parity block stream from the data block stream to decrease valid page copy during garbage collection. In the convertible programming scheme, the flash memory blocks that are allocated for parity data are programmed in SLC mode to reduce the wear caused by programming stress, while the other flash memory blocks are written in MLC mode as usual. Evaluation shows that our scheme decreased garbage collection overhead by up to 58% and improved lifespan by up to 54%, assuming that the MLC write stress was 3.5 times that of the SLC.

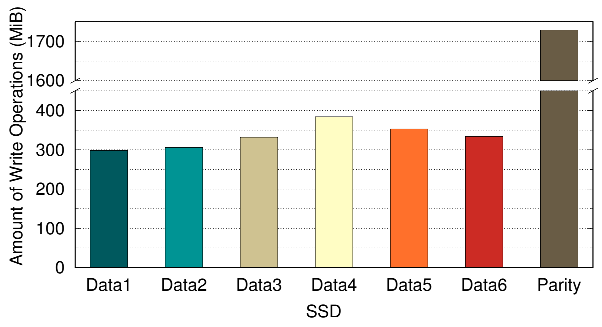


Figure 1: Number of write operations issued to each disk in a RAID-4 array during execution of a benchmark producing small randome writes.

1 Introduction

In data centers or enterprise systems, SSDs are mostly used in a form of RAID configurations for securing reliability and performance. However, the characteristic access patterns of RAID systems reduce their lifespan and thus increasing their cost-to-own (CTO) [2, 16, 19, 26].

A representative case is the well-known issue of partial-stripe writes [13, 16, 18, 19]. In a RAID system, when an arbitrary data block in a stripe is modified, the corresponding parity block must be recalculated and updated, resulting in significantly higher write traffic for parity blocks than for data blocks.

Figure 1 shows the number of write operations issued to each disk in a RAID during execution of the FIO microbenchmark [1]. The benchmark was configured to issue small random writes of a few tens of KB in size. In this experiment, to clearly separate parity block writes from data block writes, we used RAID-4 instead of RAID-5 or RAID-6.

The results show that the SSD dedicated to parity blocks received significantly higher write traffic than the other SSDs in the RAID. These frequent updates of parity blocks will significantly reduce lifespan of SSDs by heavy flash cell wear out and degrade overall write performance of RAID systems by increasing garbage collection overhead as well.

Based on this observation, we propose two RAID management schemes, parity-stream separation and single-level cell (SLC)/multi-level cell (MLC) convertible programming, to increase the performance and lifespan of a flash SSD RAID system. The parity-stream separation scheme, which utilizes multi-streamed SSDs [12], reduces the number of valid page copy operations caused from garbage collection by separating the parity-stream from the data-stream at the RAID controller level. The SLC/MLC convertible programming scheme lessens the wearing out of flash blocks to extend lifespan of SSDs by programming flash blocks allocated for the parity-stream in SLC mode.

The proposed schemes were implemented on the FlashSim flash SSD simulator [14], and evaluated using the three benchmarks: FIO, TPC-C [20], and Yahoo Cloud Serving Benchmark (YCSB) [4].

2 Our Approach

2.1 Parity-stream separation

A stream is an abstraction of SSD capacity allocation, and each stream stores a set of data having the same lifespan expectancy. As shown in Figure 2, a multi-streamed SSD allocates physical capacity to place data in a stream together and not to mix data from different streams [12].

This approach significantly improves throughput by reducing the garbage collection overhead when the life expectancy of various streams differs. Multi-stream support was standardized for the SCSI/SAS interface and is expected to be integrated in the NVMe standard.

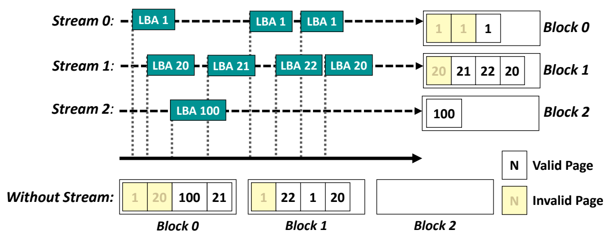


Figure 2: In this example [12], each one of the three streams writes data to its corresponding flash block. Without multi-stream support, pages will be allocated sequentially from Block 0 to Block 1.

When partial-stripe writes occur frequently, parity blocks stored in a RAID are likely to be updated more frequently than data blocks. In a RAID-5 or RAID-6, parity block updates and data block updates tend to interleave because of parity disk rotation. Therefore, the chances are high that user data pages will blend with parity pages in a single flash block. In such cases, frequently updated parity pages are likely to be invalidated soon while data pages remain valid longer. This incurs additional valid page copy operations during garbage collection. Consequently, one can find fairly good opportunities to reduce garbage collection overhead by separating parity write as an independent stream.

As shown in Figure 3, at the RAID controller level, the parity-stream separation scheme maps parity writes to parity-stream and user data writes to data-stream because the controller knows which are parity writes. Naturally, the parity writes and data writes are delivered to SSDs with their stream IDs. The FTL (flash translation layer) inside the SSDs stores parity pages in the flash blocks allocated only for the parity-stream while the data pages are stored in the flash blocks for the data-stream.

This approach is expected to significantly reduce the number of valid pages in a victim flash block for garbage collection. In addition, this will lower the write amplification factor which will end up with the improved throughput and wear out reduction of a RAID system.

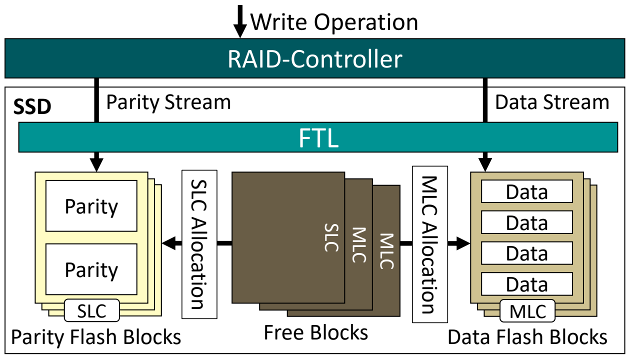


Figure 3: Parity writes are tagged as a parity-stream at the RAID controller. Flash blocks allocated to parity-stream are programmed using SLC mode.

2.2 SLC/MLC convertible programming

Although parity-stream separation suppresses valid page copy operations during garbage collection, it does not eliminate frequent parity block writes caused by partial parity updates. These frequent updates still adversely affect SSD lifespan in a RAID. Moreover, the proportion of parity block writes is expected to grow because of the expanded stripe size as the number of SSDs constituting a RAID increases to earn more parallelism. Therefore, reducing the wear caused by frequent updates is as important as reducing the garbage collection overhead. To achieve this goal, we propose SLC/MLC convertible programming.

The structure of MLC flash cells is fundamentally the same as that of SLC flash cells. The differences between SLC and MLC lie in how to program and how to interpret threshold voltages in the flash cells. Therefore, if it is possible to change its programming method and threshold voltage interpretation, a flash block can be used interchangeably in both SLC and MLC modes, depending on demand. In fact, to shorten write latency, some commercial MLC SSDs use a few of their flash blocks as a write buffer simulating high-performance SLC [21].

In general, an SLC flash cell provides a program/erase (P/E) endurance cycle more than 10 times greater than its MLC counterpart. The extended lifespan of SLC flash cells is due to low programming stress and the fewer reference voltages. Therefore, if the flash blocks dedicated to the parity-stream are managed in the SLC programming mode, wear will be significantly reduced due to lessened programming stress, and in turn, the overall lifespan of the SSD will be extended.

The proposed approach programs the flash blocks dedicated to the parity-stream in the SLC mode as shown in Figure 3. If a new free block is required for the parity-stream, the FTL first tries to allocate an existing free block that has been used in the SLC mode. If there are no such free blocks, an MLC free block will be converted to an SLC one and allocated for the parity-stream.

When an MLC flash block is used in SLC mode, the capacity will be shrunk by one half. Therefore, the available SSD capacity is decreased by programming an MLC flash block in SLC mode. In general, the commercial SSDs contain significantly greater flash memory than their nameplate capacity [22]. The size of this over-provisioned area is usually up to approximately 30% of the nameplate capacity. Decreased capacity caused by the conversion to SLC mode will be deducted from this over-provisioned area, maintaining the nameplate capacity. However, in an extreme case in which parity writes dominate, the ratio of SLC blocks will monotonically increases to the point where the nameplate capacity cannot be guaranteed. Therefore, the maximum allowable number of flash blocks that can be used in SLC mode must be enforced. When that number reaches its upper limit, even the parity-stream must be stored in the MLC mode.

This reduction of the over-provisioned capacity from the SLC conversion may increase the frequency of garbage collection, which may result in increased write amplification. Therefore, it is desirable to dynamically determine the optimal number of flash blocks to be used in SLC mode for given conditions. However, the dynamic determination model is beyond the scope of our paper and is left to future research. Instead, we analyze the trade-off between garbage collection overhead increase and wear out reduction depending on the upper limit of the SLC flash block count in Section 3.

3 Evaluation

3.1 Evaluation environment

We implemented the proposed schemes in the FlashSim SSD simulator [14] for evaluation. In addition, we added multi-stream support to the DFTL [5], which is the default FTL of FlashSim.

We used block-level I/O traces as inputs to FlashSim. We collected the block-level I/O traces using a storage server. The server used 7 SSDs, which were grouped as a RAID-5 volume, and Linux kernel 4.4.35 as its software RAID controller. A stripe of the RAID volume was 3 MB. The traces were extracted using `ftrace`. The RAID controller was modified to tag parity writes. The parity tag was delivered to the block layer and recorded in the traces together with its corresponding I/O request.

We used three benchmarks: FIO [1], TPC-C [20] on MySQL, and YCSB [4] on Cassandra. Table 1 illustrates their specific configurations.

Table 2 describes the parameters for simulating SSDs in our evaluation. The flash memory characteristics followed that of V-NAND [23].

Table 1: Workload configuration

FIO (Random write)	Runtime	20 min.
	No. of jobs	4
	Write size	16 KB
	File size	4 GB
	Distribution	Zipfian θ=0.99
TPC-C	Runtime	2 h.
	No. of warehouses	40
YCSB (Update only)	Heap size	2 GB
	No. of threads	32
	Record size	1 KB
	No. of records	4 millions
	No. of operations	3 billions
	Distribution	Zipfian θ=0.99

Table 2: Parameters used for SSD simulation

Total block number		2048
Over provisioning area		28%
Page size		16 KB
No. of pages per block	SLC	128
	MLC	256
Read delay	SLC	20 us
	MLC	35 us
Write delay	SLC	180 us
	MLC	390 us
Block erase latency		4000 us
MLC:SLC write stress		1.5×, 2.5×, 3.5×

No technical specifications reveal the difference in programming stress between SLC and MLC modes when the same flash memory is interchangeably written in both modes. However, in such cases, the ratio of programming stress per byte between the two modes reported as being from 1.4 [11] to 3.5 [3]. Accordingly, we simulated varying the ratio between 1.5 and 3.5.

3.2 Benchmark results

The number of valid page copies caused by garbage collection is the main determinant of garbage collection overhead and causes response delays when the number of free blocks available suddenly drops due to a burst write requests. Figure 4 shows the normalized number of copied pages caused by garbage collection.

Page copy operations diminished by 58% for FIO because its continual, small, random writes maximized the benefit of parity-stream separation. The average write size of TPC-C was significantly larger than that of FIO because the former used buffered I/O and the latter used

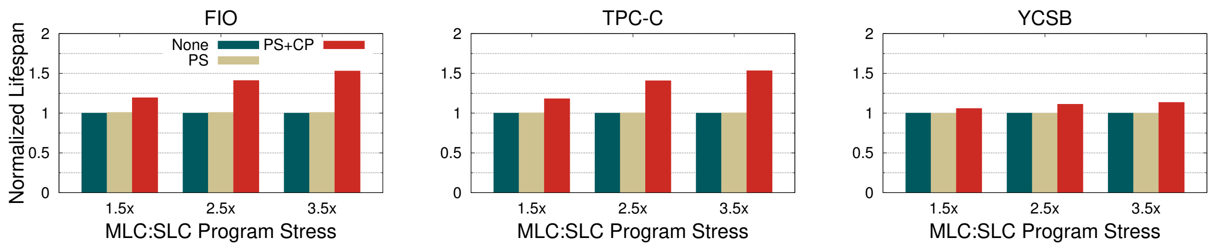


Figure 5: Normalized expected lifespans under various MLC:SLC programming stress ratios.

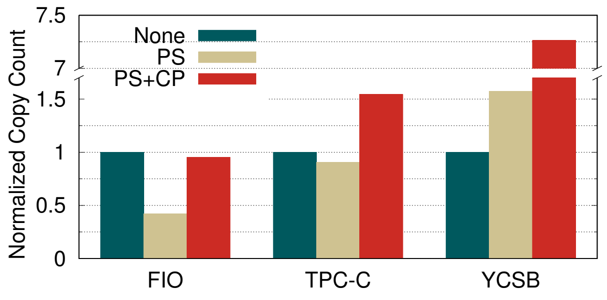


Figure 4: Normalized number of page copy operations for garbage collection. PS and CP mean parity-stream separation and convertible programming, respectively.

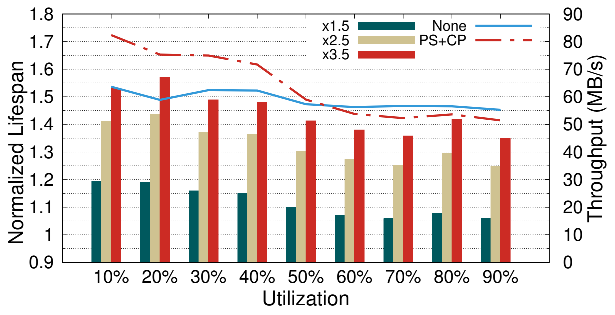


Figure 6: Throughput and lifespan measured while executing FIO with varying capacity utilization. Lines mean throughput and bars mean normalized lifespan.

direct I/O. This increased write size reduced the effectiveness of parity-stream separation.

To update its data, Cassandra first records the update in the commit log. When the log is committed, the updated data is saved in a new table file, which is immutable. Thus, every update creates a new table file, and the multiple table instances are regularly merged and compacted into a single table file [6]. Because of this copy-on-write update style, the sizes of write requests flowing from YCSB were similar to the stripe size. In addition, because our simulation did not utilize the TRIM command [9], both logging and copy-on-write updating increased the capacity utilization of simulated SSDs to close to 100%. These two factors incapacitated parity-stream separation.

As stated, convertible programming shrinks the proportion of over-provisioned capacity, resulting in increased valid copy operations for all three benchmarks. Specifically, YCSB showed an appreciably higher number of copy operations because of the extremely high capacity use mentioned. However, the actual performance decrease was smaller than the increase in copy operations because a significant percentage of copy operations was done in SLC mode.

To analyze the effect of the proposed schemes on SSD lifespans, we measured the accumulated wear out caused by programming stress during execution of benchmarks, varying the wear stress ratio between SLC and MLC modes. We found that expected lifespan was inversely proportional to accumulated wear out.

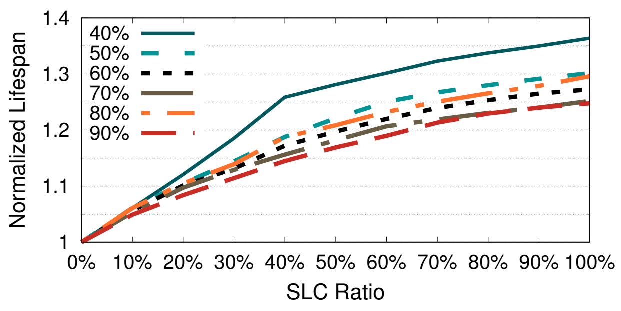
As shown in Figure 5, the convertible programming scheme appreciably extended SSD lifespans in all cases. In particular, when the stress ratio was 3.5, the proposed scheme improved the expected lifespan for TPC-C by 54%, which is a gain from the 35% wear reduction.

Because the number of copy operations for garbage collection was a relatively small portion of the overall write operations, parity-stream separation alone barely affected lifespan. For instance, in spite of increased copy operations, the YCSB lifespan was improved by approximately 10%.

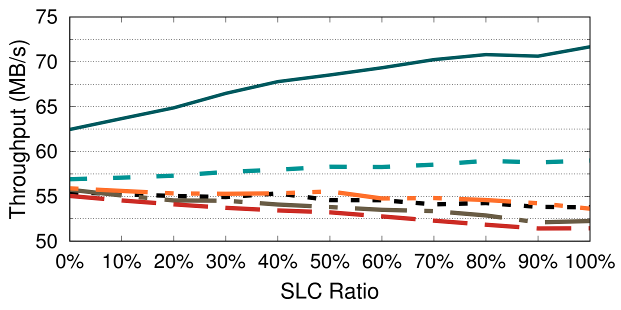
3.3 Effect of utilization and SLC ratio

As shown in the YCSB results, using SLC mode under condition of high capacity utilization shrank the relative size of over-provisioned area, and thus increasing garbage collection overhead. To investigate the relationship between the effectiveness of the proposed schemes and capacity utilization, we observed throughput and lifespan changes under our approach while varying the capacity utilization of the FIO benchmark.

Figure 6 shows that with the proposed schemes the throughput proportionally decreased to the capacity utilization. When the utilization was over 50% the throughput was poorer than that without the proposed schemes.



(a) Normalized expected lifespan



(b) Throughput

Figure 7: Lifespan and throughput measured while executing FIO with varying upper limit of SLC block ratio. The programming stress ratio was set at 2.5×. Each line represents a different capacity utilization.

However, storage capacity is being severely underutilized in most of data centers [17], and the appropriate use of the TRIM command keeps the capacity utilization at the FTL level the same as that at the file system level. Therefore, the proposed schemes will operate under favorable conditions in terms of capacity utilization most of the time.

Diminishment of lifespan improvement also occurred in proportion to capacity utilization. However, the life expectancy of SSDs was markedly improved in all conditions. In particular, when the ratio of programming stress between the MLC and SLC modes was 2.5, the proposed schemes achieved lifespan improvement of approximately, even at 90% capacity utilization.

Figure 7 presents the lifespan and throughput changes according to the upper limit of the ratio of the flash blocks that are allowed to be programmed in SLC mode. An SLC ratio of 0% means that using of SLC mode is prohibited, whereas an SLC ratio of 100% means that the size of the over-provisioned area may become zero. Since this ratio is the upper limit, naturally, the real number of blocks used in SLC mode may be lower.

Figure 7a shows that the higher the number of allowable SLC blocks the more the lifespan was increased, regardless of capacity utilization. However, the proposed schemes more effectively increased lifespan when the utilization was low. In contrast, as shown in Figure 7b, the proposed schemes decreased throughput when the utilization was greater than 50%.

4 Related Work

In order to improve reliability and extend mean time between failure (MTBF), a few parity distribution schemes for SSD RAIDs were proposed, which include uneven distribution [2] and wear-level-aware distribution [26]. These approaches allow RAIDs to sidestep the damage caused by frequent parity updates to some degree.

Postponing parity writes using non-volatile memory buffer can efficiently reduce the number of parity updates issued to storage devices [7, 8, 24]. This approach is orthogonal to our schemes and can be combined together with them to minimize write amplification due to parity updates.

An SSD management scheme that consolidates parity pages to reduce garbage collection overhead has been proposed [25]. It is technically challenging to differentiate parity pages from data pages inside an SSD without stream information. Therefore, this approach distinguishes parity pages by detecting the different access patterns from the upper RAID layer.

Dynamic SLC/MLC switching of flash cells has been used to increase SSD performance and lifespan. Worn-out MLC blocks can be revived by using them in SLC mode [10]. A file system has been proposed that divides flash memory into SLC and MLC regions and dynamically changes the size of each region to meet the changing requirements of applications [15].

5 Conclusions

This paper proposed and evaluated the parity-stream separation and SLC/MLC convertible programming to remedy the adverse effects caused by partial-stripe writes in SSD RAIDs. Evaluation showed that the proposed schemes significantly reduced garbage collection overhead when there are frequent small random writes, and improved lifespan of SSDs for all evaluation cases.

We expect that splitting the parity-stream into multiple streams based on the update patterns of their corresponding stripes will further reduce garbage collection overhead. In addition, we plan to investigate a model that dynamically determines the optimal upper bound of the number of SLC flash blocks in terms of workload and system characteristics.

6 Acknowledgement

This work was supported by Memory Business, Samsung Electronics Co., Ltd.

References

[1] AXBOE, J. Flexible IO tester. `https://github.com/axboe/fio`.

[2] BALAKRISHNAN, M., KADAV, A., PRABHAKARAN, V., AND MALKHI, D. Differential RAID: rethinking RAID for SSD reliability. In *Proceedings of the 5th European conference on Computer systems (EuroSys)* (2010), ACM, pp. 15–26.

[3] CACTUS TECHNOLOGIES. An overview of Pseudo-SLC NAND. White Paper, 2016.

[4] COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing (SoCC)* (2010), ACM, pp. 143–154.

[5] GUPTA, A., KIM, Y., AND URGAONKAR, B. DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems (ASPLOS)* (2009), ACM, pp. 229–240.

[6] HEWITT, E. The Cassandra architecture. In *Cassandra: the definitive guide*, M. Loukides, Ed. OReilly Media, Inc., 2010, ch. 5, pp. 87–98.

[7] HSIEH, J., AND SU, C. Parity management scheme for a hybrid-storage RAID. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC)* (2016), ACM, pp. 1774–1776.

[8] IM, S., AND SHIN, D. Flash-aware RAID techniques for dependable and high-performance flash memory SSD. *IEEE Transactions on Computers 60*, 1 (2011), 80–92.

[9] INTEL, CO. Intel solid-state drive optimizer. White Paper, 2009.

[10] JIMENEZ, X., NOVO, D., AND IENNE, P. Phoenix: reviving MLC blocks as SLC to extend NAND flash devices lifetime. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)* (2013), EDA Consortium, pp. 226–229.

[11] JIMENEZ, X., NOVO, D., AND IENNE, P. Libra: software-controlled cell bit-density to balance wear in NAND flash. *ACM Transactions on Embedded Computing Systems (TECS) 14*, 2 (2015), 1–22.

[12] KANG, J.-U., HYUN, J., MAENG, H., AND CHO, S. The multi-streamed solid-state drive. In *Proceedings of the 6th USENIX conference on Hot Topics in Storage and File Systems (HotStorage)* (2014), USENIX Association.

[13] KIM, T., LEE, S., PARK, J., AND KIM, J. Efficient lifetime management of SSD-based RAIDs using dedup-assisted partial stripe writes. In *2016 5th, Non-Volatile Memory Systems and Applications Symposium (NVMSA)* (2016), IEEE, pp. 1–6.

[14] KIM, Y., TAURAS, B., GUPTA, A., AND BHUVAN, U. Flash-Sim: a simulator for NAND flash-based solid-state-drives. In *Proceedings of the 2009 First International Conference on Advances in System Simulation (SIMUL)* (2009), IEEE, pp. 125–131.

[15] LEE, S., HA, K., ZHANG, K., KIM, J., AND KIM, J. FlexFS: a flexible flash file system for MLC NAND flash memory. In *Proceedings of the 2009 conference on USENIX Annual technical conference (USENIX)* (2009), USENIX Association, pp. 9–9.

[16] LI, Y., LEE, P. P., AND LUI, J. C. Analysis of reliability dynamics of SSD RAID. *IEEE Transactions on Computers 65*, 4 (2016), 1131–1144.

[17] MEARIAN, L. Survey finds storage systems underutilized as companies add more capacity. *Computer World* (July 1, 2010).

[18] MOON, S., AND REDDY, A. Don't let RAID raid the lifetime of your SSD array. In *Proceedings of the 5th USENIX conference on Hot Topics in Storage and File Systems (HotStorage)* (2013), USENIX Association, pp. 7–7.

[19] MOON, S., AND REDDY, A. Does RAID improve lifetime of SSD arrays? *ACM Transactions on Storage (TOS) 12*, 3 (2016), 11.

[20] POESS, M., AND CHRIS, F. New TPC benchmarks for decision support and web commerce. *ACM SIGMOD Record 29*, 4 (2000), 64–71.

[21] SAMSUNG ELECTRONICS, CO., LTD. Samsung solid state drive TurboWrite technology. White Paper, 2013.

[22] SAMSUNG ELECTRONICS, CO., LTD. Over-provisioning: maximize the lifetime and performance of your SSD with small effect to earn more. White Paper, 2014.

[23] SAMSUNG ELECTRONICS, CO., LTD. Samsung V-NAND technology. White Paper, 2014.

[24] WANG, M., AND HU, Y. i-RAID: a novel redundant storage architecture for improving reliability, performance, and life-span of solid-state disk systems. In *Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC)* (2016), ACM, pp. 1824–1831.

[25] WU, X., XIAO, N., LIU, F., CHEN, Z., DU, Y., AND XING, Y. RAID-Aware SSD: improving the write performance and lifespan of SSD in SSD-based RAID-5 system. In *Proceedings of the 2014 IEEE Fourth International Conference on Big Data and Cloud Computing (BDCLOUD)* (2014), IEEE, pp. 99–103.

[26] YIMO, D., FANG, L., ZHIGUANG, C., AND XIN, M. WeLe-RAID: a SSD-based RAID for system endurance and performance. In *Proceedings of the 8th IFIP International Conference on Network and Parallel Computing (NPC)* (2011), Springer-Verlag, pp. 248–262.