Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM

Eunji Lee
Chungbuk Nat'l University
Cheongju, Korea
eunji@cbnu.ac.kr

Julie Kim
Ewha University
Seoul, Korea
julie.kim@linecorp.com

Hyokyung Bahn*
Ewha University
Seoul, Korea
bahn@ewha.ac.kr

Sam H. Noh
UNIST
Ulsan, Korea
samhnoh@gmail.com

Abstract—**Write amplification is a critical factor that limits the stable performance of flash-based storage systems. To reduce write amplification, this paper presents a new technique that cooperatively manages data in flash storage and nonvolatile memory (NVM). Our scheme basically considers NVM as the cache of flash storage, but allows the original data in flash storage to be invalidated if there is a cached copy in NVM, which can temporarily serve as the original data. This scheme eliminates the copy-out operation for a substantial number of cached data, thereby enhancing garbage collection efficiency. Experimental results show that the proposed scheme reduces the copy-out overhead of garbage collection by 51.4% and decreases the standard deviation of response time by 35.4% on average.**

Keywords-Flash Memory; Write Amplification; Non-volatile Memory

I. INTRODUCTION

Developments in nonvolatile memory (NVM) technologies such as PCM (phase-change memory), STT-MRAM (spin torque transfer magnetic RAM) and 3D XPoint are advancing rapidly [1-5]. While deploying emerging nonvolatile memories requires making changes in the storage management mechanisms devised for conventional volatile memory based architectures [6], it also opens opportunities to improve on limitations that also exist. In this paper, we consider the use of NVM as a cache for flash based storage devices and propose the Cooperative Data Management (CDM) scheme that cooperatively manages data in flash and the NVM cache in order to reduce the write amplification of flash memory.

Flash memory is widely used as storage in high-end systems as well as small embedded devices. Flash memory is an erase-before-write medium and the erasure unit (called block) is much larger than the write unit (called page) [7]. Thus, an entire block needs to be erased even if a small portion of the block is updated. To alleviate this inefficiency, writes in flash memory are performed out-of-place, and space containing obsolete data is periodically recycled. This procedure, which is called garbage collection, selects blocks to be recycled, copies out valid pages in the blocks, if any, and then erases the blocks. Garbage collection incurs additional writes to flash, that is, writes are amplified. This write amplification is a critical factor that limits the stable performance of flash storage [8-15].

The key idea of the CDM scheme comes from the observation that we can recycle victim blocks in flash without copying out their valid data if the data resides in nonvolatile cache, which provides durability, instead of flash storage. This scheme exploits the non-volatility of scalable cache to relieve the wear-out of flash storage. This is an important contribution when one considers the fact that NVM density, cost, and performance is constantly improving, while the endurance of flash keeps deteriorating [16-19].

The basic workings of our scheme can be summarized in the following two situations. First, when cached data becomes dirty, our scheme immediately notifies this state change to the storage, allowing early invalidation of storage data. Second, when garbage collection is activated, our scheme erases victim blocks without copying out their valid data if the data, possibly being clean, resides in NVM. To support this mechanism, we define a new flash page state that we call "removable" and discuss how this state can be utilized when managing data in flash storage and the NVM cache.

The proposed scheme is implemented on SSDsim, which is an extended version of DiskSim for SSDs [20]. Experimental results with various storage workloads show that the proposed scheme reduces the copy-out overhead of garbage collection by 51.4% on average. This also leads to an average of 15% reduction in response time. Such reduction in write amplification also results in a 35.4 % reduction in standard deviation of the response time.

II. ANALYZING WRITE AMPLIFICATION FACTOR

The write amplification factor is defined as the amount of data actually written to flash storage over the amount of data writes that was requested. We investigate the write amplification factor of SSD by making use of SSDsim, which is a high-fidelity event-driven SSD simulator, as commercial SSD devices do not provide internal information to the end users. (Detailed configurations of the experiment will be described later in Section 4.) We observe the write amplification factor with respect to the workload characteristics by using two synthetic workloads (Random and Sequential) and two real workloads (JEDEC and OLTP). For the synthetic workloads, we generate 5 million write operations in random and sequential patterns, respectively, by making use of the internal workload generator in SSDsim [20]. Each operation is in 8 sector (4KB) units and the total footprint is 20GB. Table 1 summarizes the characteristics of the real workloads used in our experiments. JEDEC (JEDEC 219A) is a workload that is used for SSD endurance verification [21]. OLTP trace that is attained at financial institutions generates I/O accesses for financial transactions [22].

Table 1: Summary of workload characteristics.

	JEDEC	OLTP
# of ops.	5,000,000	9,034,179
Ratio of ops.	write 89%, trim 6%, flush 5%	read 52% write 48%
Footprint	31.2GB	30.7GB

Before measurements begin we warm up the simulator by writing data sequentially until the number of free blocks reduces to less than 5% of the total flash blocks. The results reported are those of repeatedly running the workload 10 times for each workload. As shown in Figure 1, the average write amplification factors of random and sequential are 2.84 and 1.00, respectively. In random workloads, as flash pages are updated at random, the victim block selected during garbage collection is likely to have many valid pages. This leads to increased write amplification. In contrast, as pages within a block are invalidated together in sequential workloads, there is no write amplification.

However, most workloads in real systems are a certain mixture of sequential and random workloads. To mimic such real situations, we generate mixed workloads and investigate the write amplification factor as the ratio of sequential and random accesses is varied. All other configurations other than the ratio of random to sequential requests are the same as that of Figure 1. Figure 2 shows the write amplification factor of a 32GB SSD as the ratio of sequential and random accesses is varied. As shown in the figure, the write amplification factor of mixed workloads is almost identical to that of a pure random workload and does not decrease even when most accesses are sequential. That is, only a small fraction of random accesses is necessary to intensify the write amplification factor, which implies that the write amplification of real workloads would be similar to that of random accesses.

This is also consistent with the write amplification factor measured from real I/O workloads as shown in Figures 1(c) and 1(d), which are very similar to that of Random in Figure 1(a). When storage capacity is small compared to the data to be stored, the write amplification factor becomes very high, going up to as much as 6.0 in our experiments. Such write amplification degrades the performance of flash storage and also reduces the lifetime of flash storage.

To solve this problem, we propose a new data management scheme that enables flash memory to minimize valid page copy-out by making use of nonvolatile cache. The next section describes the design and implementation details of the Cooperative Data Management (CDM) scheme that we propose.

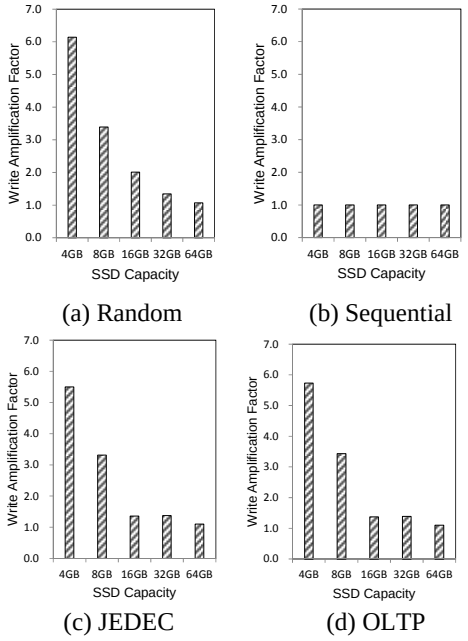


Figure 1: Write amplification factor for various workloads.

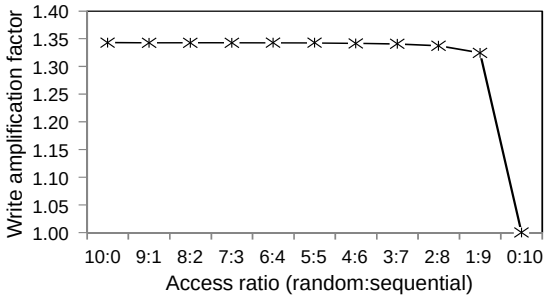


Figure 2: Write amplification factor as access ratio is varied.

III. Cooperative Data Management with Flash Memory and NVM

A. Cooperative Data Management

Traditional caching systems employ volatile media such as DRAM and thus, the original data in storage must be preserved even though there is another copy in the cache. However, when the cache becomes nonvolatile and data is cached, we now have two persistent copies in the system; one in flash, which we refer to as the *storage (or flash) copy* and the other in NVM cache, which we refer to as the *cached copy*. One of these copies can (and should) be eliminated as keeping both is redundant and thus, may be costly. Whether to discard the cache or the flash copy will depend on which benefits the system more. For example, if the storage copy is a candidate to be moved due to garbage collection, it might be better to simply discard it instead as a valid, nonvolatile cached copy exists. In this manner, the Cooperative Data Management (CDM) scheme that we propose recycles victim blocks without copying out their valid data if the data resides in the cache.

Note that the storage copy could be invalidated as soon as it is uploaded in cache. However, this will increase writes to flash as, in this case, the cached copy must be written back to flash when evicted from cache as the storage copy is now

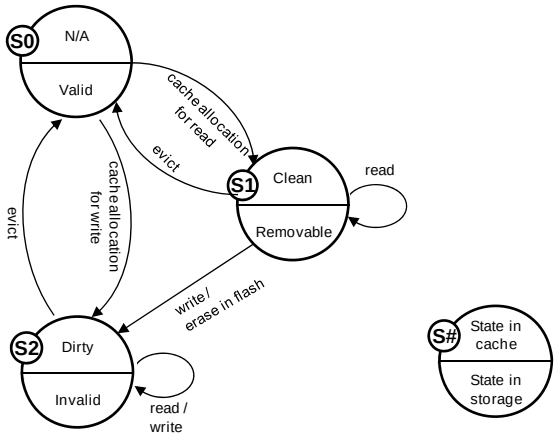


Figure 3: States of cache and storage data when the NVM cache and flash storage cooperate.

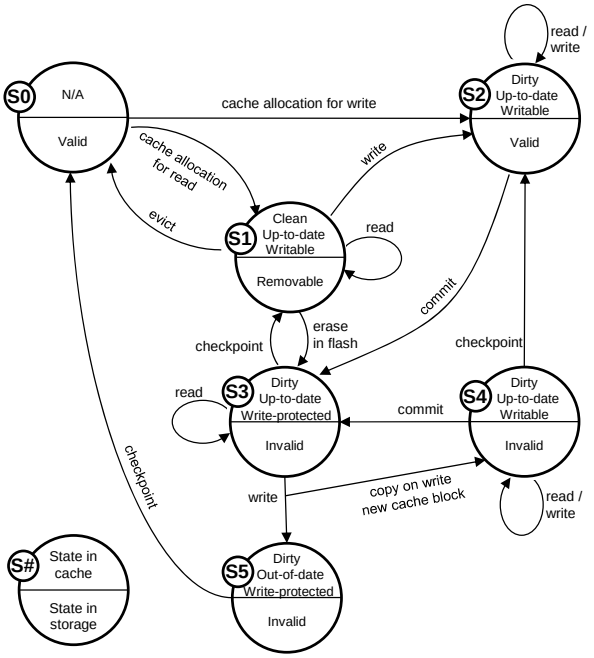


Figure 4: States of storage data and cached copy in the consistency guaranteed model.

invalid. Thus, instead of invalidating storage copy right after it is cached, we change the state from "valid" to "removable", where "removable" is a flash page state that is defined as being currently "valid" but may be removed (i.e., be considered to be "invalid") without copying it out if an erasure occurs on the storage copy. If the cached copy is evicted before the storage copy is recycled, the state of the storage copy simply returns to "valid."

On the other hand, if the cached copy is updated (i.e., becomes dirty), our scheme immediately invalidates the storage copy because the cached copy must be written back to flash anyhow. Note here that periodic flushing done in traditional volatile caches to maintain consistency becomes unnecessary as the cache is now nonvolatile.

Figure 3 shows the state diagram of data in flash (storage) and cache when the NVM cache cooperates with flash. Each circle represents the state of the storage copy and its cached copy. For simplicity, let us assume that storage and cached copies are managed in page units. The storage copy has three states, "valid," "invalid," and "removable," and the cached copy has two states "clean" and "dirty." In the figure, in state S0 the data only exists in flash. When the storage copy is retrieved and cached, the state changes to S1. Now, as the up-to-date copy exists in both cache and flash, the state of the storage copy becomes "removable." If the cached copy is updated or the storage copy is removed, the up-to-date data exists only in the cache, and the storage copy becomes "invalid" (S2). Finally, if the cached copy in the "dirty" state is evicted, it is flushed to flash and the state returns to S0. If a cached copy in the "clean" state (S1) is replaced, it is simply discarded from the cache without flushing (S0).

B. Consistency Issues

Management of cached data requires a guaranteed level of reliability. For example, modern reliable file systems perform out-of-place updates such as journaling (e.g., Ext4 and ReiserFS) or copy-on-write (e.g., BtrFS and ZFS) to support recovery of file systems to the latest consistent state. A certain level of reliability is also necessary when using our CDM scheme. Here, we discuss the consistency mechanism for CDM.

Recall that unlike flash memory, in-place updates are possible with NVM but atomicity of in-place updates are guaranteed in only relatively small sizes [23, 24]. Now suppose that only the cached copy of a certain page remains. In this situation, if the system crashes while updating the cached copy, the data may become inconsistent as page overwrite is not atomic. To overcome this problem, we prohibit in-place updates of cached copies when it serves as the original data.

Figure 4 extends the states defined in Section 3.A to guarantee consistency. We define two additional indicators "writable/write-protected" and "up-to-date/out-of-date." The writable/write-protected indicator distinguishes whether the cached copy allows in-place update or not. If a write is requested on a write-protected copy, it is first copied and the write is performed on that copy. The up-to-date/out-of-date indicator distinguishes whether the cached copy is the most recent version or not. This is necessary as multiple copies for the same data may exist in the cache.

Let us see how our scheme works with the state changes. Initially, data exists only in flash (S0). Upon read/write requests, the data is cached and S0 transits to S1/S2. In S1 state, as up-to-date data exists in both cache and flash, the storage copy becomes "removable." In this situation, suppose that garbage collection occurs and the storage copy needs to be erased. Then, the cached copy becomes "write-protected" before the storage copy becomes "invalid" (S3). This protects the cached copy from being corrupted upon a crash. The data

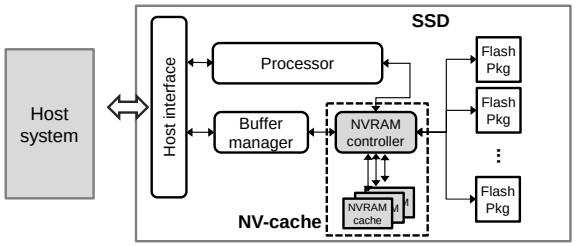


Figure 5: System architecture used in our experiments.

in this state still services read requests, but upon a write request, a copy-on-write occurs to protect the original data (S5) and the update is performed on a new cache location (S4). The out-of-date copy (S5) is maintained until its up-to-date version (S4) is successfully committed, and then reclaimed (S0).

Returning to state S1, if the cached copy is updated while its original data exists in flash, the storage copy becomes out-of-date. However, as the cached copy has not yet committed, the storage copy is still in the "valid" state (S2). This is different from the basic model presented in Section 3.A that immediately changes the out-of-date data to "invalid", but does not guarantee consistency.

Similar to file system journaling that periodically commits updated data to a separate storage area (e.g., every 5 seconds), the proposed scheme can periodically perform commits by changing the state of cached "dirty" data (S2/S4) to "write-protected" (S3). Note that this procedure just changes the state of the cached copy, incurring neither storage writes nor copy operations within the cache. Once the data is committed, it becomes the new original data and the data before the update needs to be invalidated. This old data may exist either in flash or in cache. If the storage copy is invalidated (S2→S3), unnecessary copy overhead during garbage collection can be eliminated. If the cached copy is invalidated (S5→S0), it is reclaimed and becomes free. The committed up-to-date data (S3) are finally reflected to a permanent storage location via checkpointing. Checkpointing should be triggered when free memory drops below low watermark. Even if the committed data (S3) becomes out-of-date (S5) due to subsequent writes, it (S5) is written to the storage during checkpointing as the up-to-date copy (S4) has not yet been committed. After checkpointing, the committed obsolete data (S5) is reclaimed (S0) and the up-to-date copy (S4) transits its state (S2) as it now has a backup copy in storage. The up-to-date cached copy (S3 or S4) still serves as a cache block (S1 or S2) after checkpointing.

C. *Implementation Issues*

The scheme that we propose can be deployed at two different levels in real systems. The first is using NVM as a host-side cache for flash based primary storage. In this case, our scheme can be supported by the kernel through the modification of the file system buffer cache layer. However, in this case, current storage interfaces need to be revised. This is because the host OS and the storage system must be able to notify each other of state changes that occur in the cache and storage system. One way to get around this limitation is to extend the host interface to transfer the state information [25]. Fortunately, this is becoming a viable approach as emerging storage interfaces like NVMe or Universal PCI Express are flexible such that adding proprietary extensions are becoming feasible [26].

The other level is to use NVM as an internal write buffer in flash storage devices such as SSDs. In this case, our scheme can be incorporated into the design of FTL in SSD internals without modifying host interfaces. This approach does not require any modifications to the storage stack as the FTL is a purely internal mechanism within the flash storage device. Even though the scheme that we propose can be deployed at both levels as just discussed, for our performance evaluation, we will only focus on the latter case.

Table 2: Experimental parameters.

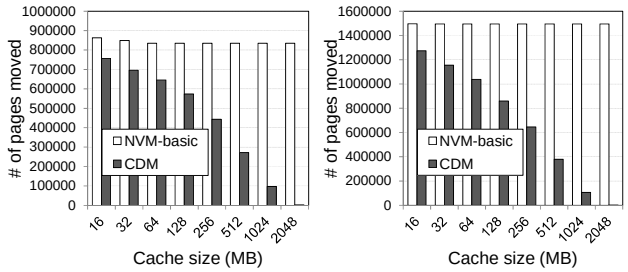
SSD capacity	64GB
Page size	4KB
Block size	256KB (64 pages)
Page read latency	25us
Page write latency	200us
Block erase latency	1.5ms
Data transfer latency	100us (for 4KB page)
Overprovisioning ratio	15%

IV. PERFORMANCE EVALUATION

For our evaluation, we implement the proposed scheme into DiskSim's MSR SSD extension [20]. The SSD simulator emulates SLC NAND flash memory chip operations, and the parameters that we use are presented in Table 2. In all configurations, there are 8 flash memory chips and the total storage capacity is 64GB. The simulator assumes the PCI-e interface with 8 lanes with 8b/10b encoding, providing 2.0 Gbps per lane.

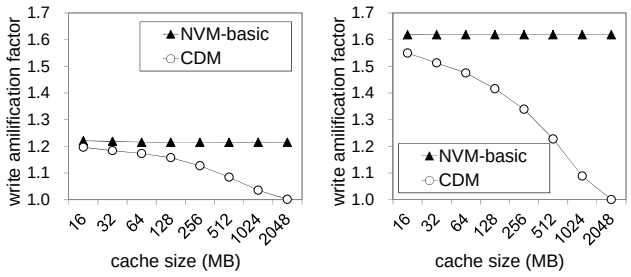
In this SSD simulator, we add a nonvolatile cache and implement the CDM scheme within the FTL. The cache is managed in 4KB page units using the LRU replacement policy. Specifically, we add the "removable" state to flash pages in conjunction with the "valid/invalid" states by modifying the page table entries in FTL. We then revise the garbage collector to skip copy-out operations for pages with the "removable" state. Before erasing blocks, the garbage collector determines that the pages in removable states are to be erased and sets their states to write-protected. Figure 5 shows the system architecture of the proposed scheme. For all experiments, we warm up the simulator in the same manner used for the write amplification experiments.

We compare our scheme with NVM-basic, which uses the same NVM cache architecture, thus providing durability against power failures, but does not perform cooperative data management. Figure 6 shows the number of pages copied out



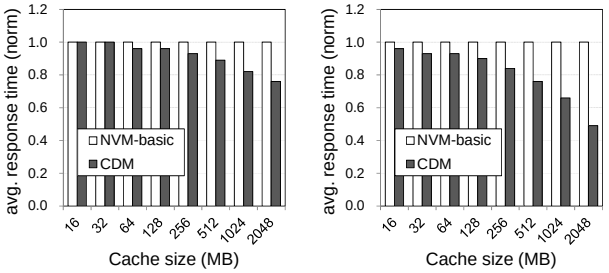
(a) JEDEC (b) OLTP

Figure 6: Number of pages copied out.



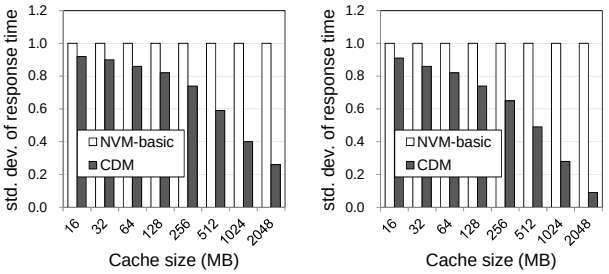
(a) JEDEC (b) OLTP

Figure 7: Write amplification factor.



(a) JEDEC (b) OLTP

Figure 8: Average response time.



(a) JEDEC (b) OLTP

Figure 9: Standard deviation of response time.

during garbage collection as a function of the cache size. As shown in the figure, the proposed scheme significantly reduces the copy-out overhead. Specifically, the improvement becomes larger as the cache size increases. This is because a large size cache allows for aggressive invalidation of copies in flash. The average reduction of copied pages is 48.3% and 54.4% for JEDEC and OLTP, respectively, compared to the original system. This will eventually lead to prolonging the lifetime of flash memory.

Figure 7 shows the write amplification factor as a function of the cache size. As shown in the figure, the proposed scheme significantly reduces the write amplification factor, specifically when the cache size becomes large. The reduced write amplification factor is in the range of 2.1-17.6% and 4.3-38.2% for the JEDEC and OLTP workloads, respectively. Figures 8 and 9 show the average response times and their standard deviation for CDM normalized to NVM-basic. Due to the large reduction in garbage collection overhead, CDM improves the average response time by 9.7% and 20.3% and reduces the average standard deviation by 31% and 39% for JEDEC and OLTP, respectively.

V. Related Works

Lu et al. observe that the layered file system and FTL design accelerates flash memory wear-out as it prevents file systems from exploiting the characteristics of flash storage devices [27]. To remedy this deficiency, they propose an object-based flash translation layer (OFTL) to which the file system storage management component is offloaded so that flash memory can be managed directly. Kang et al. propose a multi-streamed I/O mechanism where the host explicitly informs storage the lifetime of the data being transferred such that the storage device can manage data more efficiently with respect to garbage collection [9]. Robert proposes dynamic overprovisioning methods for storage systems through compression and deduplication of data for reduction of write amplification and increased endurance and longevity [10]. Skourtis et al. propose redundant data management and separation of reads and writes to avoid unpredictable delays caused by garbage collection [11]. Jagmohan et al. propose a multi-write coding mechanism that enables a NAND flash page to be programmed more than once without block erase, thereby relieving write amplification by garbage collection [12]. Boboila and Desnoyers present a method using actual chip-level measurements to reverse engineer FTL details [13]. They show that performance and endurance can be estimated through the reverse engineering FTL. This method is used to suggest FTL parameter setting such that performance and endurance can be efficiently balanced. Yang et al. present analytic modeling for evaluating write amplification in garbage collection and reveal the relationship between endurance and performance metric [14].

VI. Conclusion

This paper presented a new data management scheme for flash storage when NVM is adopted as the cache. The proposed scheme cooperatively manages data in flash and the cache in order to efficiently perform garbage collection. Specifically, we allow victim blocks to be erased without copying out their valid data if the data are in the cache. Experimental results show that the proposed scheme reduces the copy-out overhead of garbage collection by 51.4% on average. This results in reduced write amplification, which in

turn, results in reduced response time and variance of response time.

VII. Acknowledgement

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2014R1A1A3053505 and No. 2015R1A2A2A05027651) and by the IT R&D program MKE/KEIT (No. 10041608). Hyokyung Bahn is the corresponding author of this paper.

References

[1] J. C. Mogul, E. Argollo, M. Shah, and P. Faraboschi, “Operating system support for NVM+DRAM hybrid main memory,” Proceedings of USENIX Workshop on Hot Topics in Operating Systems (HotOS), 2009.

[2] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, “Scalable high performance main memory system using phase-change memory technology,” Proceedings of the 36th International Symposium on Computer Architecture (ISCA), 2009.

[3] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Phase change memory architecture and the quest for scalability,” Communications of the ACM, 53(7), 2010.

[4] S. Lee, H. Bahn, and S. H. Noh, “CLOCK-DWF: a write-history-aware page replacement algorithm for hybrid PCM and DRAM memory architectures,” IEEE Transactions on Computers, 63(9), pp. 2187-2200, 2014.

[5] https://en.wikipedia.org/wiki/3D_XPoint

[6] E. Lee, H. Bahn, and S. H. Noh, “Unioning of the buffer cache and journaling layers with non-volatile memory,” Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST), pp. 73-80, 2013.

[7] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, “A space-efficient flash translation layer for compact flash systems,” IEEE Transactions on Consumer Electronics, 48(2), 2002.

[8] Y. Lu, J. Shu, and W. Zheng, “Extending the Lifetime of Flash-based Storage through Reducing Write Amplification from File Systems,” Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST), pp. 73-80, 2013.

[9] J. Kang, J. Hyun, H. Maeng, and S. Cho, “The Multi-streamed Solid-State Drive,” Proceedings of the 6th USENIX Workshop on Hot Topics in Storage and File systems (HotStorage), 2014.

[10] Robert L. Horn, “Dynamic overprovisioning for data storage systems”, Western Digital Technologies, 2015.

[11] D. Skourtis, D. Achlioptas, N. Watkins, C. Maltzahn, and S. Brandt, “Flash on Rails: Consistent Flash Performance through Redundancy,” Proceedings of the USENIX Annual Technical Conference (ATC), 2014.

[12] A. Jagmohan, M. Franceschini, L. Lastras, “Write Amplification Reduction in NAND Flash through Multi-Write Coding,” Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2010.

[13] S. Boboila and P. Desnoyers, “Write Endurance in Flash Drives: Measurements and Analysis,” Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST), 2010

[14] Y. Yang , J. Zhu, “Algebraic modeling of write amplification in hotness-aware SSD,” Proceedings of the 8th ACM International Systems and Storage Conference (SYSTOR), 2015.

[15] P. Desnoyers, “Analytic modeling of SSD write performance,” Proceedings of the 5th ACM International Systems and Storage Conference (SYSTOR), 2012.

[16] M. Yang, Y. Chang, C. Tsao, and P. Huang, “New ERA: new efficient reliability-aware wear leveling for endurance enhancement of flash storage devices,” Proceedings of the 50th Annual Design Automation Conference (DAC), 2013.

[17] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, “Spin-transfer torque magnetic random access memory (STT-MRAM),” ACM Journal on Emerging Technologies in Computing Systems, 9(2), 2013.

[18] O. Zilberberg, S. Weiss, and S. Toledo, “Phase-change memory: An architectural perspective,” ACM Computing Surveys, 45(3), 2013.

[19] Y. Li and K. N. Quader, “NAND Flash memory: challenges and opportunities,” Computer, pp. 23-29, 2013.

[20] N. Agrawal, V. Prabhakaran, T. Wobber, J. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” Proc. USENIX ATC, pp. 57-70, 2008.

[21] JEDEC, Master trace for 128 GB SSD, http://www.jedec.org/standards-documents/docs/ jesd219a_mt.

[22] UMASS trace repository, http://traces.cs.umass.edu.

[23] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger and D. Coetzee, “Better I/O through byte-addressable, persistent memory,” Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP), pp.133-146, 2009.

[24] S. R. Dulloor, S. Kumar, A. Keshavamurthy, P. Lantz, D. Reddy, and R. Sankaran, J. Jackson, “System software for persistent memory,” Proceedings of the 9th European Conference on Computer Systems (EuroSys), 2014.

[25] F. Shu, “Data set management commands proposal for ATA8-ACS2,” T13 Technical Committee, United States: At Attachment:e07154r1, 2007.

[26] A. Huffman, “NVM Express: Going Mainstream and What’s Next”, Intel Developers Forum, 2014.