

# Hexa: Overcoming Capacitance Constraints in SSDs

Anonymous

**Abstract**—The growth in SSD capacity is reaching its limit due to the stunted growth of capacitors—electrical components that store charge to protect data for the volatile memory in case of power loss. This paper presents **Hexa**, a novel SSD-internal DRAM management scheme that allows the SSD capacity to scale beyond the slow growth of capacitors. **Hexa** suppresses an increase of the dirty memory footprint within buffer using the deep queues available in today’s storage interfaces. We implement our design in **FEMU**, an open-source SSD development framework and demonstrate that **Hexa** delivers IOPS close to 90% of performance at only 1% of capacitance compared to the existing scheme.

**Index Terms**—component, formatting, style, styling, insert

## I. INTRODUCTION

The enterprise-class SSDs adopt the capacitor to protect data durability in case of power crash. This technique is called Power-Loss Protection (PLP) [11, 18, 23] and it is needed because SSDs use a DRAM as an internal buffer for absorbing user writes and caching translation information (also known as mapping table). If they are not protected, SSDs will have not only a data loss and/or corruption but also a long recovery time to build an up-to-date mapping table by scanning entire flash drives. To preclude this situation, the enterprise-class SSDs rely on the capacitors that reserves energy to safely persist data of the volatile buffer in a power loss.

However, the heavy reliance on capacitors is no longer sustainable as the increase in SSD far outpaces the increase in capacitor density. The SSD has increased significantly in density for the past decade. In 2011, a typical 2.5-inch SSD had 256GB capacity, but by 2018, a high-capacity SSD boasted a 30TB, expanding by 100× over the past ten years [1, 21]. This remarkable growth of the device-capacity is thanks to the advanced scaling technologies such as nanoscale fabrication [3] and multi-layer stacking [20]. Al(aluminum) and Ta(tantalum)-electrolytic capacitors used in SSDs have increased in density by tenfold from 1960 to 2005. This is approximately 50x slower than the SSD density increase rate. Given that the internal buffer size increases in proportion to the storage capacity, the slow scaling of capacitors will eventually limit the amount of DRAM that can be used in an SSD. This, in turn, will also limit the storage capacity as the size of DRAM and aggregate flash capacity proportionally scale [19, 22].

This paper presents **Hexa**, a novel SSD-internal DRAM management scheme that allows the SSD capacity to scale beyond the slow growth of capacitors. SSD-internal DRAM is used for (1) caching translation information (also known as mapping table) and (2) buffering user writes. In typical SSD designs, most of the DRAM is used for caching the mapping table and the buffer for user writes is kept at a minimal

(just enough to hide the flash program latency) [13]. As an example, Samsung PM1633a 15.36TB SSD houses 16GB DRAM [1]. Given that the mapping table size is typically 0.1% of storage capacity [19, 22], we can assume that only 4% of DRAM is used for a write buffer.

As opposed to a memory pressure, the negative impact of data loss is equally serious for both data. Because an SSD writes the associated LPN (Logical Page Number) in the OOB (Out-of-band) area of the physical page, it is virtually possible to recover the up-to-date mapping table by scanning the entire NAND flash memory. However, because it takes prohibitively long, particularly for the scalable SSDs, PLP-SSD snapshots an entire mapping table into the specific area in NAND flash in a power loss and loads it into DRAM at a reboot. The user data also offers no alternative but for PLP as it cannot be recovered after a crash. For the PLP-SSD, the host system ensures reliability assuming that all acknowledged data survive a power outage, and thus, the loss of user data can lead to a catastrophic result.

With these properties in mind, we invent an in-device buffer management mechanism for SSD under capacitance constraints. **Hexa** partially protects the mapping table, while fully protecting user data under capacitance constraints. If the number of dirty pages in mapping table goes beyond the limit, changes are immediately flushed to NAND flash. Instead, **Hexa** buffers more user writes so that mapping entry eviction becomes more efficient by aggregating dirty updates. This substantially reduces the amount of mapping table-related write traffic, and in turn, improves the overall performance under capacitance constraints.

**Hexa** is built upon the current trend of increasing the queue depth of the storage interfaces. SATA and SAS support a single queue with 32 and 245 commands, but NVMe has up to 65,535 queues with as many as 65,536 commands per queue. This extension allows SSDs to further optimize the internal activities by taking advantage of the outstanding request information.

We implement **Hexa** in **FEMU**, an open-source SSD development framework [17]. The performance evaluation with

SSD Model	Manufacturer	Class	PLP	Capacitor
950Pro, 850Pro	Samsung	Client	None	-
M500	Micron	Client	Partial	Ceramic
M500DC	Micron	Enterprise	Full	Tantalum
PM863, SM863	Samsung	Enterprise	Full	Tantalum
DC1000B	Kingston	Enterprise	Full	Tantalum
DC S3700, S3500	Intel	Enterprise	Full	Aluminum

TABLE I: **Power Loss Protection in SSDs [11, 18, 23].** **EJ:** This table is also included in ISLPED paper. Should we take it out?

various workloads shows that Hexa offers 82% and 94% of IOPS of the full-protection SSD when a protected ratio is 1% and 10%, while a conventional SSD provides 69% and 81% of performance.

## II. RELATED WORK

The need for reducing the energy consumption needed for power-loss protection arises in different contexts. A few studies reduce the total energy consumption by speeding up the back-up process at a power failure using the fast media. Guo et al. reduce the capacitance requirement by writing back the volatile buffer data into PRAM (Phase Change Random Access Memory), which is faster and uses lower power than NAND flash [7]. They argue that this reduction enables to replace the supercapacitors that are suffering from serious aging problems with the regular capacitors, which have more reliable characteristics [9]. This consequently enhances the robustness of storage device. As a similar approach, Smartbackup [10] proposes dynamic NAND channel allocation and SLC (single-level cell) mode programs to make the dump process shorter at sudden power-off. It makes full use of available SSD channels and dynamically adjusts these channels based on the available power of the capacitor to exploit the nature of high parallelism on NAND flash arrays. In addition, as the SLC mode program shows significantly shorter time than subsequent MLC, TLC, or QLC mode, it programs the target page to dump in SLC mode to achieve shorter time required for dumping process.

Another approach to reducing the capacitor size is protecting a part of the volatile buffer. DRWB (Dual-Region Write Buffer) divides the internal-SSD buffer into small protected region (backed by a capacitor) and large unprotected region and when the data on unprotected region is updated, the delta for the page is logged in the protected region [14]. With this differential logging, DRWB logically realizes the non-volatile buffer using a small size of capacitor. However, the proposed technique only regards the user data, having no consideration on the metadata such as mapping table, despite that it actually accounts for most of the internal buffer of SSDs. Furthermore, commercial SSDs typically do not cache read data in the buffer because the host memory can serve as a cache memory of the storage device. For these reasons, the effectiveness of DRWB may be limited in practical environment.

Some studies explore ways of using the internal write buffer efficiently in scalable SSDs. Chen et al. project that even the high capacity of SSDs will use the small size of write buffer because the capacitor that protects the buffer does not scale well due to the cost, size, and reliability constraints [4]. Nevertheless, they observe that the small sized write buffer can be effective for reducing write traffic in particular applications that perform journaling heavily. Motivated by this observation, they present the application-SSD co-design to reduce the data writes buffered for heavy logging/journaling applications. They propose to protect write-hot log/journal data with capacitors while the log/journal data being durable. In addition, they propose NVMe interface extension for host to notify SSDs the ranges of write-hot LBAs for more efficient

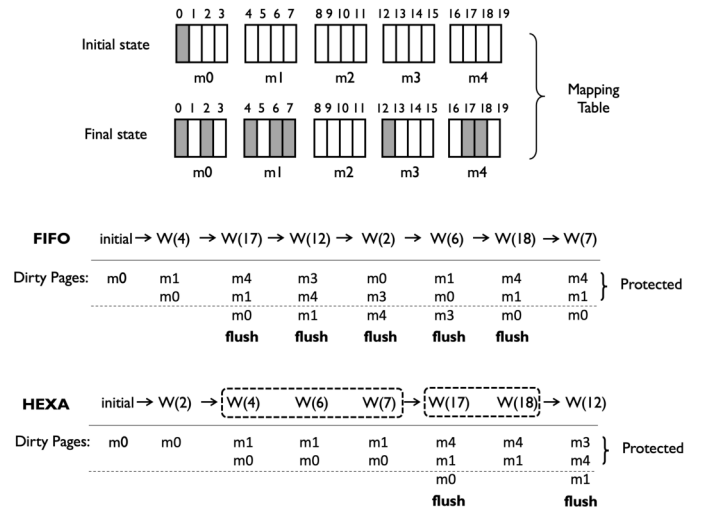


Fig. 1: Hexa-SSD buffer management scheme

protection by capacitors with reduced complexity of hot/cold separation. It reduced substantial amount of flash memory write traffic with few megabytes of capacitor-powered write buffer, but it is specific to heavy log/journal applications and requires change of application code to benefit from its scheme.

SpartanSSD [16], which is most related with our work, pinpoints capacitance constraints in scalable SSDs and reduces capacitance requirements by making use of elastic journaling. Spartan-SSD logs the mapping information updates into the in-device journal so that the writes to the mapping table can be buffered. They use a hybrid journal that is backed by a small size of DRAM and flash memory. This hybrid journal is highly flexible in terms of capacity, and thus, it enables a timely checkpoint that reflects a log data to the mapping table and flushes dirty map pages into NAND flash chips. Although Spartan-SSD also reduces translation-related writes under capacitance constraints, it not only have double write for mapping information updates, but also it essentially increases a recovery time, which could be highly harmful when the multiple SSDs are running simultaneously.

## III. DESIGN

Hexa partially protects the mapping table with limited capacitance. When the dirty pages of mapping table become more than the maximum number of protected pages, Hexa flushes them to flash memory based on the LRU (Least-recently Used) algorithm. Because this flush operation does not arise with SSD using PLP, mitigating the effect of this overhead is a key strategy to achieving high performance under capacitance constraints. To this end, Hexa presents a cost-effective scheduling scheme for the in-storage buffer. Hexa prefers to force the user data that increases the dirtiness of the mapping table the least to flash memory. This scheme reduces the dirty page footprint of the mapping table at a time window by enhancing the locality of updates. As a result, the frequency of flush operation for the mapping table can be largely reduced.

Figure 1 compares the flush overhead of FIFO and Hexa scheduling in SSD buffer. In this example, there are seven write requests in the device queue, sent from host in the following order:  $W(4)$ ,  $W(17)$ ,  $W(12)$ ,  $W(2)$ ,  $W(6)$ ,  $W(18)$ , and  $W(7)$ . The mapping table has one dirty page ( $m0$ ) at an initial state. We assume that 2 out of 5 pages of the mapping table are protected. FIFO writes the user data in the buffer to flash memory in arrival order. With this scheme, the mapping table would be randomly updated, generating a large number of dirty pages at a time window. Consequently, FIFO incurs a total of five flushes of the mapping table page during the write process.

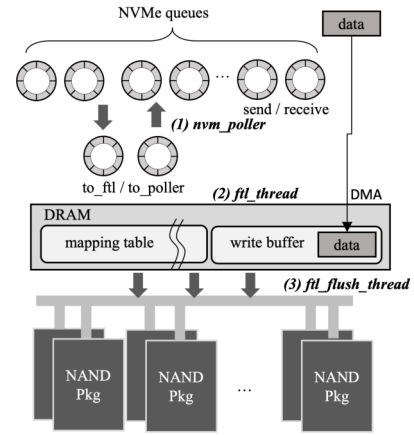
In contrast, Hexa calculates the write cost for each data that indicates an increase in the number of dirty pages of the mapping table when it is flushed, and it processes the request with minimum cost first. In this example, the write request  $W(2)$  has a top priority because its associated mapping table page ( $m0$ ) is already dirty, and thus it does not add the dirty pages of the mapping table. Next, the write requests  $W(4)$ ,  $W(6)$ , and  $W(7)$  are processed. Because their address mapping entries are located in the same page of the mapping table, the cost of flushing them is reduced to one third. With this scheme, Hexa can reduce the footprint of mapping table updates within time intervals, thereby delivering only two flushes of the mapping table for the same task.

#### IV. IMPLEMENTATION

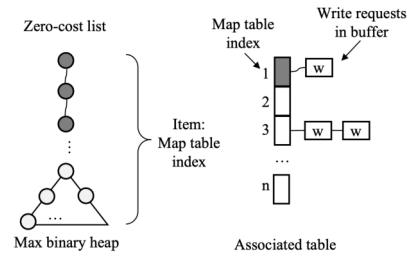
We implement Hexa in FEMU, an open-source SSD development framework [17]. Fig. 2 shows the overall architecture of Hexa-SSD and its internal data structures. As the original version of FEMU directly writes data to flash memories without write buffering, we extend it to use a small-sized write buffer, which aggregates and batches user writes into the underlying flash memory.

Hexa-SSD maintains three different threads that are executing concurrently within SSDs. The `nvm_poller` takes a charge of transferring requests between NVMe queues and FTL-internal queues. The FTL-internal queue consists of a pair of sub-queues, each of which is named `to_ftl` and `to_poller`. This separation is intended to enable a non-blocking access to queues by allowing only a single writer for each queue. Second, the `ftl_thread` essentially handles the ingress requests from the internal queues. For write, it transfers data from the host memory to the SSD-internal write buffer with DMA and updates the associated entry in a translation page to point to the write buffer. Then, it notifies the completion of request to the `nvm_poller` by enqueueing the acknowledgement into the `to_poller` queue. Because Hexa protects the entire space of write buffer with capacitance, data persistency is guaranteed for all acknowledged writes. For read, the `ftl_thread` retrieves the requested data by consulting the mapping table and transfers it to the host.

The `ftl_flush_thread` plays a role of writing data from a DRAM-buffer into a flash memory. With the FIFO policy, the user writes are issued to NAND flash memory in the order they arrive into the buffer. However, Hexa flushes



(a) Architecture



(b) Data structures for FTL

Fig. 2: Hexa-SSD Internals.

buffered writes in the order such that it least increases the dirty memory footprint of the mapping table. To realize this design, Hexa maintains two data structures, as depicted in Fig. 2(b). First, a *zero-cost list* that holds the indexes to translation pages that is already in a dirty state, and second, a *max binary heap* that maintains the indexes to translation pages sorted by the number of buffered user write requests associated with that page.

When a half of the write buffer becomes occupied, flushing is invoked. Hexa-SSD first flushes user data whose translation pages in the zero-cost list, and then persists user data as their translation pages are ordered by the max binary heap. By doing so, each user write minimizes the number of eventual translation page write, and each translation page write maximizes the number of persisted mapping entries. These data structures are updated by the `ftl_thread` when a write request arrives at SSD. To exploit the SSD internal parallelism, we send data to flash memory in batches by the number of NAND flash chips that can be written simultaneously.

Once the write operations of NAND flash memory complete, `ftl_flush_thread` updates the mapping table entries to point to the physical address of the data in a flash memory. At this moment, if the number of dirty mapping table pages goes beyond the protectable number of pages, `ftl_flush_thread` persists the mapping table page to flash memory. This is also conducted in batches by the number

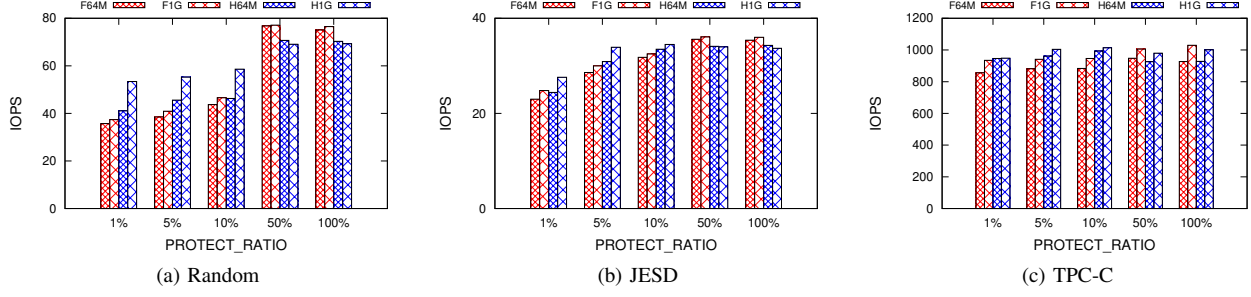


Fig. 3: IOPS: $F$  and  $H$  denotes  $FIFO$  and  $HEXA$ .

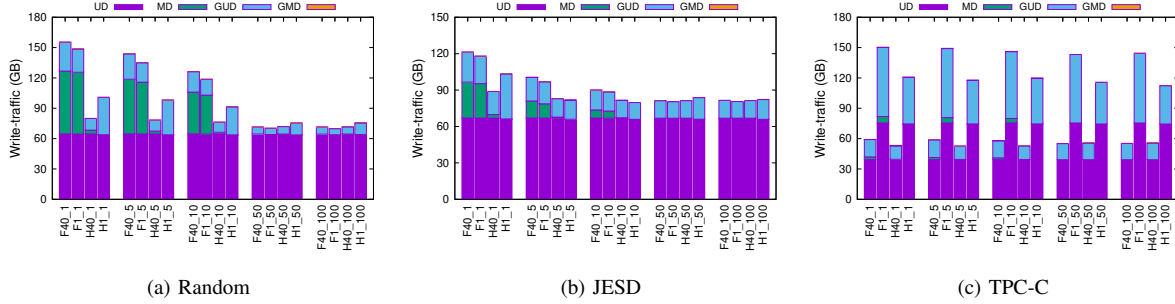


Fig. 4: **Write Traffic.**  $UD$  - User Data,  $MD$  - Mapping Data,  $GUD$  - GC Write for User Data,  $GMD$  - GC Write for Mapping Data.

of NAND flash chips that can be written in parallel.

## V. EVALUATION

We perform the experiments on a machine with a 20-core Intel Xeon(R) Silver 4114 CPU running at 2.2GHz and 84GB memory. We run FEMU (QEMU-based SSD emulator) configured to use 10 cores, 4GB DRAM for main memory, and 16GB DRAM for SSD emulation. The SSD maintains a mapping table entirely in DRAM partially protected with capacitance. The NAND flash chips include 8 channels and 8 flash LUNs per channel. The page size is 8KB and the per-block pages are 256. The read and write latency is set to 60us and 700us, respectively [5]. We use the greedy algorithm for GC(Garbage-Collection), which selects the least utilized block as a victim for cleaning. The Ext4 file system is mounted on the emulated SSD.

We measure the average IOPS and the write traffic varying the protected ratio of a mapping table from 1% to 100%. We study two different sizes of write buffer, 64MB and 1GB, to investigate the effectiveness of Hexa with respect to a queue depth. The performance evaluation is conducted using three workloads. The fio benchmark [2] generates the 4KB of random writes and the skewed read-write mixed workload that follows JESD219 using 4 threads. A total of 64GB of data was written to the 4GB area. For the real workload, we use TPC-C [6] on MySQL, an online transactional processing benchmark, which is executed using a sysbench benchmark suite [24]. The TPC-C preconditions an SSD with data writes

for 300 seconds and generates write queries for 180 seconds using 10 threads. For the performance comparison, we also implemented an FIFO-SSD, which uses a FIFO(First-In-First-Out) scheduling policy for processing write requests within SSDs.

Fig.3 show the IOPS of FIFO-SSD and Hexa-SSD with different size of write buffers, and Fig. 4 shows the write traffic breakdown for the scenario. When the protected ratio is less than 10%, Hexa has improved the performance by up to XX% compared to the existing FIFO version. Compared with full protection SSD, when the protected ratio is 1%, FIFO performance degradation occurs more than XX%-XX%, whereas in Hexa, performance degradation is XX% - XX%. This performance enhancement is achieved by significantly reducing the cost of flushing overflow dirty pages to the SSD through the in-buffer re-ordering.

One counter-intuitive result is that Hexa has slightly lower IOPS than FIFO when protected ratio is above 50%. Our careful analysis reveals that the reordering of Hexa distorts the original write pattern generated by the host, which increases the possibility that pages with different lifetimes are stored in the same block. As a result, this increases the number of valid pages that need to be copied from the GC, thereby increasing the write traffic of the GC. Even when the workload is synthetic random, the host writes have temporal locality because they are transferred through a file system. Therefore, distortion of host write pattern through re-ordering causes degradation of GC performance. Note that the target environment of this

paper is a case where the protected ratio is low, but in order to improve the generality of Hexa, we will study the technique that can alleviate the above problem in the future.

## VI. CONCLUSION

In this paper, we raised an issue about capacitance constraints in scalable SSDs and presented a novel SSD design called Hexa to overcome the limitation. Hexa-SSD protects a part of the buffer, but reduces the dirty memory footprint by exploiting the increasing queue depth of the storage interfaces. We implemented a Hexa-SSD prototype in FEMU, an open-source SSD development framework. Performance evaluation using the prototype shows that Hexa-SSD delivers only XX%-XX% performance slowdown when capacitance is reduced to 1%, while conventional SSD using FIFO decreases performance by XX%.

## REFERENCES

- [1] AnandTech. Samsung 30.72 TB SSDs: Mass production of PM1643 begins. <https://www.anandtech.com/show/12448/samsung-begins-mass-production-of-pm1643-sas-ssds-with-3072-tb-capacity>, 2018.
- [2] Jens Axboe. fio - flexible i/o tester. <https://github.com/axboe/fio>, 2021.
- [3] Christoph Busche, Laia Vilà-Nadal, Jun Yan, Haralampos N Miras, De-Liang Long, Vihar P Georgiev, Asen Asenov, Rasmus H Pedersen, Nikolaj Gadegaard, Muhammad M Mirza, et al. Design and fabrication of memory devices based on nanoscale polyoxometalate clusters. *Nature*, 515(7528):545–549, 2014.
- [4] Xubin Chen, Yin Li, and Tong Zhang. Reducing flash memory write traffic by exploiting a few MBs of capacitor-powered write buffer inside solid-state drives (ssds). *IEEE Trans. Computers*, 68(3):426–439, 2019. <https://doi.org/10.1109/TC.2018.2871683>.
- [5] Woosong Cheong, Chanho Yoon, Seonghoon Woo, Kyuwook Han, Daehyun Kim, Chulseung Lee, Youra Choi, Shine Kim, Dongku Kang, Geunyeong Yu, et al. A flash memory controller for 15 $\mu$ s ultra-low-latency ssd using high-speed 3d nand flash with 3 $\mu$ s read time. In *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 338–340. IEEE, 2018.
- [6] Transaction Processing Performance Council. Tpc benchmark c standard specification, 1990.
- [7] Jie Guo, Jun Yang, Youtao Zhang, and Yiran Chen. Low cost power failure protection for MLC NAND flash storage systems with PRAM/DRAM hybrid buffer. In *Design, Automation and Test in Europe, DATE 13, Grenoble, France, March 18-22, 2013*, pages 859–864, 2013. <https://doi.org/10.7873/DATE.2013.181>.
- [8] Aayush Gupta, Youngjae Kim, and Bhuvan Ugaonkar. Dftl: a flash translation layer employing demand-based selective caching of page-level address mappings. *Acm Sigplan Notices*, 44(3):229–240, 2009.
- [9] Jiaoying Huang, Liang Mei, and Cheng Gao. Life prediction of tantalum capacitor based on gray theory optimization model. In *2011 IEEE International Conference on Quality and Reliability*, pages 166–171. IEEE, 2011.
- [10] Min Huang, Yi Wang, Liyan Qiao, Duo Liu, and Zili Shao. SmartBackup: An efficient and reliable backup strategy for solid state drives with backup capacitors. In *17th IEEE International Conference on High Performance Computing and Communications, HPCC 2015, 7th IEEE International Symposium on Cyberspace Safety and Security, CSS 2015, and 12th IEEE International Conference on Embedded Software and Systems, ICSS 2015, New York, NY, USA, August 24-26, 2015*, pages 746–751, 2015. <https://doi.org/10.1109/HPCC-CSS-ICSS.2015.180>.
- [11] Intel. Power loss imminent(PLI) technology. <https://www.intel.com/content/www/us/en/solid-state-drives/ssd-power-loss-imminent-technology-brief.html>, 2014.
- [12] Song Jiang, Lei Zhang, XinHao Yuan, Hao Hu, and Yu Chen. S-FTL: An efficient address translation for flash memory by exploiting spatial locality. In *2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–12. IEEE, 2011.
- [13] Woon-Hak Kang, Sang-Won Lee, Bongki Moon, Yang-Suk Kee, and Moonwook Oh. Durable write cache in flash memory SSD for relational and nosql databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 529–540, 2014. <https://doi.org/10.1145/2588555.2595632>.
- [14] Dongwook Kim and Sooyong Kang. Dual region write buffering: making large-scale nonvolatile buffer using small capacitor in SSD. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 2039–2046, 2015. <https://doi.org/10.1145/2695664.2695830>.
- [15] Hyukjoong Kim, Dongkun Shin, Yun Ho Jeong, and Kyung Ho Kim. SHRD: Improving spatial locality in flash storage accesses by sequentializing in host and randomizing in device. In *15th USENIX Conference on File and Storage Technologies (FAST)*, pages 271–284, 2017.
- [16] Hyeon Gyu Lee, Juwon Lee, Minwook Kim, Donghwa Shin, Sungjin Lee, Bryan S Kim, Eunji Lee, and Sang Lyul Min. Spartanssd: a reliable ssd under capacitance constraints. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2021.
- [17] Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Matias Björling, and Haryadi S Gunawi. The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator. In *16th USENIX Conference on File and Storage Technologies (FAST)*, pages 83–90, 2018.
- [18] Micron. How Micron SSDs handle unex-

- pected power loss. [https://www.micron.com/-/media/client/global/documents/products/white-paper/ssd\\_power\\_loss\\_protection\\_whitepaper10.pdf](https://www.micron.com/-/media/client/global/documents/products/white-paper/ssd_power_loss_protection_whitepaper10.pdf), 2014.
- [19] Fan Ni, Chunyi Liu, Yang Wang, Chengzhong Xu, Xiao Zhang, and Song Jiang. A hash-based space-efficient page-level FTL for large-capacity SSDs. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–6. IEEE, 2017.
  - [20] Jae-Woo Park, Doogon Kim, Sunghwa Ok, Jaebeom Park, Taeheui Kwon, Hyunsoo Lee, Sungmook Lim, Sun-Young Jung, Hyeongjin Choi, Taikyung Kang, Gwan Park, Chul-Woo Yang, Jeong-Gil Choi, Gwihan Ko, Jaehyeon Shin, Ingon Yang, Junghoon Nam, Hyeokchan Sohn, Seok-In Hong, Yohan Jeong, Sung-Wook Choi, Changwoon Choi, Hyun-Soo Shin, Junyoun Lim, Dongkyu Youn, Sanghyuk Nam, Juyeab Lee, Myungkyu Ahn, Hoseok Lee, Seungpil Lee, Jongmin Park, Kichang Gwon, Woopyo Jeong, Jungdal Choi, Jinkook Kim, and Kyo-Won Jin. 30.1 a 176-stacked 512gb 3b/cell 3d-nand flash with 10.8gb/mm<sup>2</sup> density with a peripheral circuit under cell array architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 422–423, 2021.
  - [21] Samsung. Samsung SSD 830 series. <https://www.samsung.com/us/support/owners/product/128gb-ssd-830-series>, 2011.
  - [22] Samsung. Samsung V-NAND SSD 860 QVO. [https://www.samsung.com/semiconductor/global.semi.static/Samsung%20SSD%20860%20QVO%20Data%20Sheet\\_Rev1.pdf](https://www.samsung.com/semiconductor/global.semi.static/Samsung%20SSD%20860%20QVO%20Data%20Sheet_Rev1.pdf), 2013.
  - [23] Samsung. Power loss protection in ssds - how ssds are protecting data integrity. A Samsung Electronics White Paper, 2016.
  - [24] Sysbench. Sysbench. <https://github.com/akopytov/sysbench.git>, 2022.