Managing Array of SSDs When the Storage Device is No Longer the Performance Bottleneck

Byungseok Kim Jaeho Kim Sam H. Noh

UNIST
(Ulsan National Institute of Science and Technology)

1 Introduction

With the advent of high performing NVMe SSDs, the bottleneck of system performance is shifting away from the traditional storage device. In particular, the I/O stack software layers have already been recognized as a heavy burden on the overall I/O. Efforts to alleviate this burden have been considered [1, 2, 3, 4]. Recently, the spotlight has been on the CPU. With computing capacity as well as the means to get the data to the processor now being limited, recent studies have suggested that processing power be pushed into where the data is residing [5, 6, 7, 8]. With devices such as 3D XPoint [9, 10, 11] in the horizon, this phenomenon is expected to be aggravated.

In this paper, we focus on another component related to such changes. In particular, it has been observed that the bandwidth of the network that connects clients to storage servers is now being surpassed by storage bandwidth [12, 13]. Figure 1 shows the changes that are happening. We observe that the changes in the storage interface is allowing storage bandwidth to surpass that of the network. As shown in Table 1, recent developments in SSDs have resulted in individual SSDs providing read and write bandwidth in the 5GB/s and 3GB/s range, respectively, which surpasses or is close to that of 10/25/40GbE (Gigabit Ethernet) that comprise the majority of networks being supported today.

Based on this observation, in this paper, we revisit the organization of disk arrays. Specifically, we target write performance in all-flash arrays, which we interchangeably refer to as SSD arrays, that are emerging as a solution for high-end storage [14, 15, 16, 17, 18, 19, 20]. As shown in Table 2, most major storage vendors carry such a solution and these products employ plenty of SSDs to achieve large capacity and high performance [16, 17, 18, 19]. Figure 2 shows how typical all-flash arrays would be connected to the network and the host. Our goal is to provide high, sustained, and consistent write performance in such a storage environment.

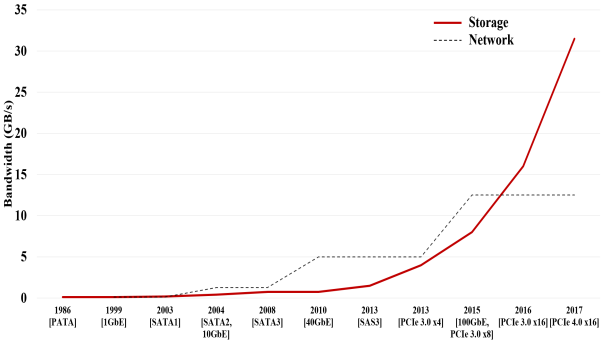


Figure 1: Network and storage bandwidth growth trend

Even though SSD arrays employ a large number of SSDs, reports have shown that contrary to expectations, they do not provide high, stable performance. In contrast, large latency variations and oscillating bandwidth for I/O requests are commonly reported [21, 22, 23, 24, 25]. The major source of such ineffectiveness is the effect of garbage collection (GC) operations within the SSD devices [26, 27].

Figure 3 shows our preliminary experimental results where we observe the write bandwidth of a RAID-0 configured SSD array comprising four identical 400GB capacity PCIe SSDs (with specifications of 128KB sequential read and write bandwidth of 2.2GB/s and 900MB/s, respectively) running the synthetic FIO workload [28]. The measurements are made at the storage host, without going through the network, and the numbers reported are averages of 5 executions of the same workload with

Table 1: Comparison of commodity SSDs

	Product	SEQ R/W (GB/s)	RND R/W (IOPS)	Interface
Intel	P3608	5/3	0.8M/0.15M	PCIe
	P3710	0.5/0.5	85K/45K	SATA
Samsung	PM1725a	6.4/3	1M/170K	PCIe
	PM1633a	1.2/0.9	190K/30K	SAS

Table 2: All-flash array products

	Solid Fire (NetApp)	EMC	Pure Storage	Nimble Storage
Model	SF19210	6X-Brick	M70	AF9000
Capacity	20TB	240TB	136TB	500TB
# of SSDs	10	150	68	96
IOPS (Random)	100K -	900K @ 8KB	370K @ 32KB	350K -
Network	20Gb	240Gb	40Gb	40Gb

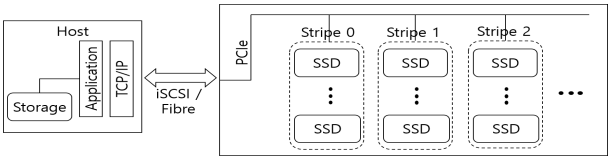


Figure 2: Organization of a typical network connected all-flash array

measurements starting with clean empty SSDs (cleaned for every execution). We observe that, at first, the performance is at its peak at roughly 3.5GB/s, which is quite close to the ideal 3.6GB/s write bandwidth. However, after some time, performance drops considerably, dropping even below the line along the x-axis at 1.25GB/s, which is the bandwidth of today's typical 10GbE network. This drop is due to GC. After the performance drop, performance stabilizes at low bandwidth and then increases, but still oscillates considerably.

We observe that with just four SSDs the storage system bandwidth can be considerably higher than typical network bandwidth. However, we also observe that due to fluctuations in SSD performance due to GC, performance can drop considerably. In conclusion, we need to find a way to stabilize performance such that services are provided in a steady and expected manner.

Efforts have been made to alleviate such performance variations at various layers of the system [27, 29, 30, 31, 32, 33]. Despite such effort, performance instability due to GC still remains a problem since GC is a necessary operation for flash memory based storage devices.

In this paper, we present a storage organization for SSD arrays that totally eliminates garbage collection (GC) within an SSD (that is, internal GC does not occur). This is based on two key, simple observations: 1) host-end storage systems are connected through the network, and 2) the storage device is no longer the bottleneck, but rather, the network bandwidth is. These two observations imply that network bandwidth performance should be the target performance goal for SSD arrays. We show that this can be achieved, not as occasional peak performance, but as sustained, consistent performance as we eliminate all internal GC.

The remainder of this paper is organized as follows. In Section 2, we discuss the organization that we propose.

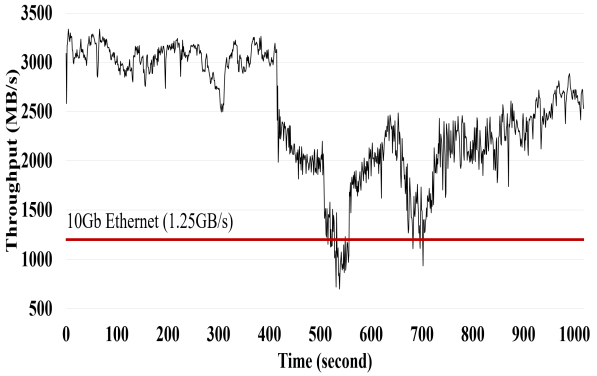


Figure 3: Observed bandwidth serving the FIO benchmark workload on a RAID-0 configured SSD array of 4 PCIe SSDs

Then, in Section 3, we discuss the experimental setup as well as the results of the measurements. Finally, we conclude the paper with Section 4.

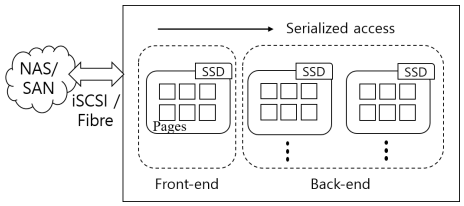
2 Serial Management of SSD Arrays

In the days of HDDs, where disk bandwidth was limited, it was logical to organize disks as an array so that individual disks could be accessed in parallel to increase bandwidth. However, with contemporary SSDs, as individual device bandwidth start to surpass network bandwidth, parallel organization of many SSDs may no longer be advantageous.

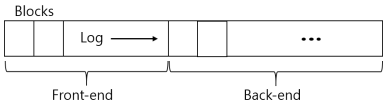
Based on this observation, we propose to organize the array of SSDs in a serial manner. This organization allows only one SSD to service all write requests, though SSDs in the array will take turns as the write servicing SSD. As we will show later, such management allows clients to observe consistent SSD write bandwidth performance. The goal of such an organization is to provide, at all times, 1) consistent unfluctuating SSD write performance, 2) at peak network bandwidth performance. This can be achieved by forbidding GC within SSDs. We now discuss the organization in detail.

2.1 Design

In describing our technique, let us for now assume that the ideal SSD write bandwidth as specified on the product is greater than the network bandwidth. This will generally be true with contemporary high-end SSDs and network configurations as we discussed earlier (for example, 3 GB/s for PCIe SSDs shown in Table 1 versus 1.25 GB/s for 10GbE). If this is not the case, we can increase the number of the so-called front-end SSDs, which we describe later, to serve our purpose. For simplicity, but without loss of generality, let us assume that we have one front-end SSD satisfying the above assumption and concentrate on writes. In this case, the network will always be the bottleneck if the SSD can be used at its maximum.



(a) The physical organization



(b) The logical view

Figure 4: Overall architecture of proposed organization

This is what we strive for. Let us assume that we have n SSDs, where n is the number of SSDs in an all-flash array, for example, as specified in Table 2.

Given these assumptions, the array of SSDs in our proposed organization can be viewed as a serial sequence of SSDs, where data is written to only one SSD at a time. As the SSD bandwidth is higher than the network bandwidth, there is no loss of performance as one SSD can absorb all write requests. We refer to the SSD serving the write request as the front-end, while the rest of the SSDs are referred to as the back-end as depicted in Figure 4(a). Logically, such an organization can be viewed as a sequential list of blocks as depicted in Figure 4(b). However, physically, each SSD is controlled independently, which is different from a typical log-structured layout.

Let us now see how we can manage the SSDs to achieve consistent, maximal performance. Assume at first that all SSDs are empty. As writes come in, all data are written only in log-structured manner without any GC. Hence, the SSD can be written with maximum performance. However, the front-end SSD will eventually fill up with a mixture of valid and invalid flash pages. Then, the front-end is replaced with a new, clean SSD. (Initially, the selected SSDs will be empty. However, with time, the selected SSD will contain a mix of valid and invalid data.) Evidently, if this keeps going, all the SSDs will fill up. Our past experience from log-structured designs tell us that some form of garbage collection is required.

Given such a scenario, let us now discuss how our proposed organization is different from the traditional log-structured approach.

- First, only the front-end SSD is actively performing writes. All back-end SSDs are free to serve read requests if there are any targeted towards these SSDs (Figure 5(a)).
- Second, no GC occurs within the frond-end SSD so that performance is maximized.

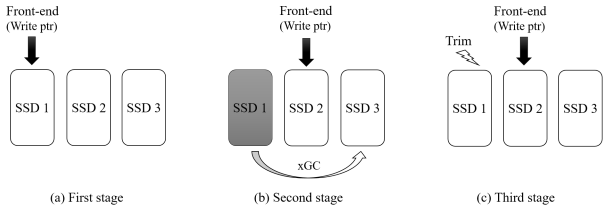


Figure 5: Sequence for handling write requests with xGC

- Third, when the front-end SSD fills up, it relieves its role as a front-end SSD to one of the back-end SSDs (Figure 5(b)).
- Fourth, GC of an SSD occurs not within SSDs, but only among back-end SSDs. Hence, we refer to this GC as external GC (xGC) in contrast to the traditional internal GC that happens within an SSD. Figures 5(b) and (c) show the sequence in which GC occurs. Once the front-end SSD relieves its role to a back-end SSD, the new front-end absorbs the writes, while the valid data in the old front-end SSD are moved to one of the back-end SSDs as depicted in Figure 5(b). When all valid data is moved, then the old front-end is cleaned by issuing a TRIM command to the entire SSD (Figure 5(c)). This makes GC a deterministic and simple activity.

2.2 Benefits of Serial Organization

There are two major benefits to our approach. The first is on performance and other is on the efficiency of the SSD. We discuss each of these below.

The performance benefits of serial management is as follows, which we quantify in Section 3.

Writes: As writes are being done in a log-structured manner, that is, a sequential manner, and no GC occurs performance is optimized.

Reads: As writes to the front-end SSD are of recent data, most of the reads targeted towards the front-end SSD will be absorbed by the cache in either the client and/or the host without affecting the front-end SSD. Similarly, reads targeted towards the back-end SSDs will not be affected by the front-end SSD write activities. Furthermore, as read latency is greatly affected by GC, and as we rid of internal GC, read performance is stabilized.

The second benefit offered by serial management is the simplicity that it brings to the design of the FTL (Flash Translation Layer) within the SSD. Recall that most high-end commodity SSDs employ large amounts of DRAM as buffer space and require substantial spare flash memory to be used as OPS (Over Provisioning Space) [34]. In addition, for high performance, techniques such as page-level mapping must be employed

over simpler, resource thrifty block-level mapping within the FTL. These kinds of restrictions impose a burden on the SSD controller in terms of cost and performance. As there is no need to perform internal GC (though in practice, we believe some form of minimal internal GC may be needed), serial management alleviates these burden on the FTL in the following manner (though these aspects are not quantified in this study):

1. Less internal resources are required for SSDs used in our organization. This is because block-level mapping, with large block sizes, may be adopted as write requests always arrive in append only manner. Hence, less memory space is required to manage the blocks within the FTL.
2. SSD capacity increases, in effect, as with no internal (or minimal) GC, no (or less) OPS space is needed to perform GC.
3. Longer lifetime can be expected for SSDs used in our organization. This is because with internal GC the same valid page may be copied multiple times within an SSD if it, by chance, happens to be in the victim block multiple times. In contrast, with our technique a valid page is written once and never copied within an SSD as there is no internal GC.
4. Wear-leveling becomes simple and even for all blocks with serially managed SSDs resulting in more predictable SSD lifetime management. This is so as an erasure happens only when a TRIM command is issued to the entire SSD. Hence, all blocks wear out evenly for the entire lifetime of the SSD.

2.3 Previous Work on Serial Management

Most previous studies on an array of disks have attempted to increases access parallelism such as in RAID configurations. In contrast, the Gecko study by Shin et al. takes a similar approach as ours in that they consider serial management of storage devices [35]. Gecko views the chain of HDDs as a log with new writes being made to the tail of the log. This is, in essence, similar to the traditional log-structured approach, and the general differences between the traditional log-structured approach and ours was described in Section 2.1.

Some other specific differences are as follows. Gecko was designed for HDD, while ours is for high-end SSDs that have bandwidth similar or surpassing that of the network. Gecko does not forbid the replacement of HDDs with SSDs, but then, does not consider the peculiarities of SSDs. In particular, our technique completely separates GC writes from first-class writes allowing first-class writes to make full use of the bandwidth. Gecko, on the other hand, intermixes them, which incurs contention at the disk that holds the tail. Furthermore, GC writes in Gecko are for segment cleaning and is different from SSD internal GC, which is not controlled in Gecko. Hence, the effect of internal GC, which is the key performance distracting factor and which our technique is obviating, still remains with Gecko.

One similarity between the two is how metadata is managed. Similarly to Gecko, we maintain a logical to physical address map as well as an inverse map in memory. Such mappings are kept for each SSD, and they occupy around 0.2% of the total capacity of the SSDs.

3 Experimental Environment and Results

In this section, we discuss the initial implementation of our proposed serial management scheme. We present the experimental setup and the benchmarks used in our evaluations, and then discuss the results.

3.1 Experimental Setup

Experimental Platform: For the storage server, we use a Dell R730 equipped with a Xeon E5-2609 CPU, 64GB DRAM, and four Intel 750 400GB NVMe SSDs. Depending on the experiment target, the four SSDs are configured in parallel manner as a volume of RAID-0 or in the serial manner that we propose resulting in 1.6TB storage capacity. The RAID-0 volume with a chunk size of 64KB is created using LVM2 [36]. For the host system, we use an x86 compatible PC with an i5-6600k CPU, 16GB DRAM, and local storage. The host is directly connected to the storage server via a 10Gb/s Ethernet card. The operating systems for the storage server and the host are Linux kernel-4.4.43 and kernel-4.3.3, respectively, and the Ext4 file system is used.

Benchmark Parameters: Two types of workloads are used for our experiments. In Section 3.2, we use a simple FTP application to isolate the effect of the network. For this workload, we make use of 10 threads with each thread transmitting a 10 GB file.

In Section 3.3, where the stability of performance is considered, we use the synthetic FIO benchmark workload. The benchmark is executed with one thread and the thread issues random writes with a queue depth of 32 and to 10 files. Over the entire experiments, we make use of two different FIO workloads. The first is the aging workload that writes for 15 minutes, where the block size is 256KB with a footprint of 1200GB. After aging, we use a second workload where 64KB sized random writes are issued for 30 minutes on a 200GB footprint.

3.2 Network as a Bottleneck

In this section, we observe how the network affects the performance of the parallel configuration, that is, RAID-0 versus the serial configuration that we propose. For both cases, we connect the storage array to the host via 10Gb/s network connection and have 10 threads in the host individually send 10GB files to write to storage via

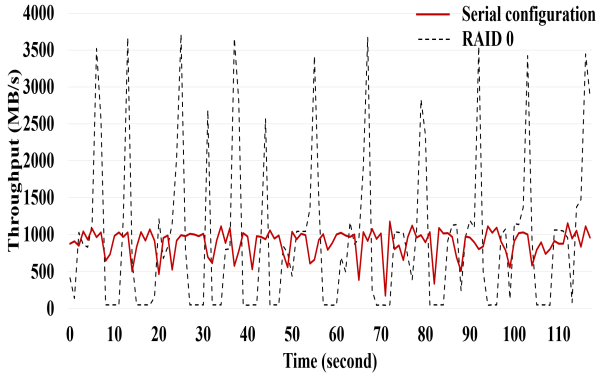


Figure 6: Performance of RAID-0 and serially configured storage connected to the host via 10GbE network

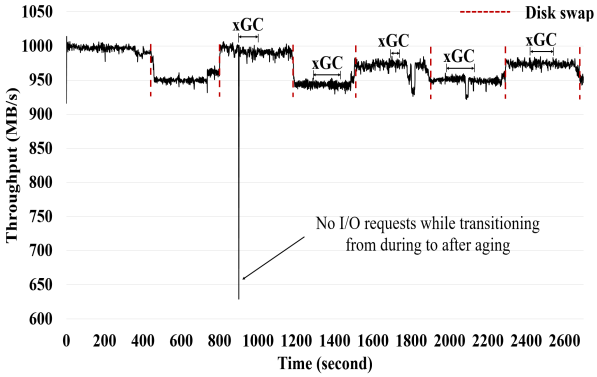


Figure 7: Proposed organization performance results

the network. We observe the throughput at the storage layer as writing a total of 100GBs saturates the network and storage bandwidth.

Figure 6 is what we observe at the storage end for RAID-0 and the serial configuration that we propose with the *y*-axis being the throughput in MB/s. We see that for RAID-0 (the dotted line) the bandwidth fluctuates considerably, periodically reaching the peak and zero values. This phenomenon occurs because as the data arrives through the network, RAID-0 processes the requests in bulks; once that is done, it has to wait for the new data to arrive through the network resulting in zero bandwidth. However, the results reveal that on average, 900MB/s throughput is maintained at the storage layer. This is below the perfect available network bandwidth, but very close to the practical peak bandwidth.

In contrast, for our serial configuration (red solid line), we also observe fluctuations but at a much smaller scale than the RAID-0 case. We find that the average bandwidth is around 890MB/s for this case, which is slightly below the 900MB/s we saw for RAID-0. However, recall that for RAID-0 all four SSDs are simultaneously being used whereas for our proposed configuration only one is being used. We see that matching the storage bandwidth with the network bandwidth is a logical choice. While this was not possible with past disk technology, with current high-end SSDs, this is easily achieved.

3.3 Sustained, Consistent Performance

We showed in the previous section that the network is the bottleneck with high bandwidth SSDs. In this section, we remove the network aspect, which is the source of performance fluctuation in the previous experiments, and concentrate on storage performance. Recall from Figure 3 how RAID-0 performance drops after some time due to GC operations. We conduct a similar experiment with our proposed serial configuration on a locally connected storage end with the workload and setting as described in Section 3.1. The average throughput results obtained for 5 measurements are shown in Figure 7. The line in the middle demarcates the results during and after aging. The demarcating line represents a sudden drop in throughput that is due to a slight pause in generating the workload during the experiments.

Overall, we see that performance during and after aging is consistent. Some points of interest in Figure 7 are as follows. First, we see that there are slight differences in the bandwidth observed when front-end and back-end SSDs are swapped as a new front-end is selected. This is because most commercial SSDs do not come with exactly the same performance. We find that performance changes according to the performance characteristics of each SSD selected as the front-end. This, we find, is consistent for each of the SSDs. Second, we see that xGC does not affect performance. Even though GC is happening, this has no bearing on performance. Finally, we attain overall high, sustained performance that we set out to attain even though we do observe occasional small dips in performance.

4 Conclusion

Based on the observation that storages devices are no longer the performance bottleneck, but that the network is, this paper proposes a serial organization for SSD arrays. Such serial organization allowed us to completely avoid internal garbage collection of SSDs such that high performance of write requests can be sustained. Experimental results showed that this is a promising approach. We are currently looking into specific details concerning the implementation of this approach and other possible design issues that we might have overlooked. To consider the scalability issue, we also plan to implement a prototype all-flash array with many more SSDs.

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. 2015R1A2A2A05027651).

References

[1] Adrian M. Caulfield, Arup De, Joel Coburn, Todor I. Mollow, Rajesh K. Gupta, and Steven Swanson. Moneta: A High-Performance Storage Array Architecture for Next-Generation, Non-volatile Memories. In *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '10, pages 385–395, 2010.

[2] Jisoo Yang, Dave B. Minturn, and Frank Hady. When Poll is Better Than Interrupt. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 1–7, 2012.

[3] Steven Swanson and Adrian M. Caulfield. Refactor, Reduce, Recycle: Restructuring the I/O Stack for the Future of Storage. *IEEE Computer*, 46(8):52–59, August 2013.

[4] Michael Wei, Matias Bjorling, and Philippe Bonnet. I/O Speculation for the Microsecond Era. In *Proceedings of the USENIX Annual Technical Conference*, ATC'14, pages 475–481, 2014.

[5] Sangyeun Cho, Chanik Park, Hyunok Oh, Sungchan Kim, Youngmin Yi, and Gregory R. Ganger. Active Disk Meets Flash: A Case for Intelligent SSDs. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 91–102, 2013.

[6] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. Query Processing on Smart SSDs: Opportunities and Challenges. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1221–1230, 2013.

[7] Boncheol Gu, Andre S. Yoon, Duck-Ho Bae, Insoon Jo, Jinyoung Lee, Jonghyun Yoon, Jeong-Uk Kang, Moonsang Kwon, Chanho Yoon, Sangyeun Cho, Jaeheon Jeong, and Duckhyun Chang. Biscuit: A Framework for Near-data Processing of Big Data Workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 153–165, 2016.

[8] Sudharsan Seshadri, Mark Gahagan, Sundaram Bhaskaran, Trevor Bunker, Arup De, Yanqin Jin, Yang Liu, and Steven Swanson. Willow: A User-programmable SSD. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, pages 67–80, 2014.

[9] Intel. Intel Optane Technology. `http://www.intel.com/content/www/us/en/architecture-and-technology/intel-optane-technology.html/`.

[10] Rick Merritt. 3D XPoint Steps Into the Light. `http://www.eetimes.com/document.asp?doc_id=1328682`, 2016.

[11] Ian Cutress. Intel's 140GB Optane 3D XPoint PCIe SSD Spotted at IDF. `http://www.anandtech.com/show/10604/intels-140gb-optane-3d-xpoint-pcie-ssd-spotted-at-idf`, 2016.

[12] Rob Davis. The Network is the New Storage Bottleneck. `https://www.datanami.com/2016/11/10/network-new-storage-bottleneck/`, 2016.

[13] Gunna Marripudi and llker Cebeli. How Networking Affects Flash Storage Systems. Flash Memory Summit 2016.

[14] Mingzhe Hao, Gokul Soundararajan, Deepak Kenchammana-Hosekote, Andrew A. Chien, and Haryadi S. Gunawi. The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies*, FAST'16, pages 263–276, 2016.

[15] John Colgrove, John D. Davis, John Hayes, Ethan L. Miller, Cary Sandvig, Russell Sears, Ari Tamches, Neil Vachharajani, and Feng Wang. Purity: Building Fast, Highly-Available Enterprise Flash Storage from Commodity Components. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'15, pages 1683–1694, 2015.

[16] EMC. EMC All-Flash Array. `https://www.emc.com/collateral/data-sheet/h12451-xtremio-4-system-specifications-ss.pdf`.

[17] PureStorage. Purestorage All-Flash Array. `https://www.purestorage.com/content/dam/purestorage/pdf/datasheets/ps_ds5p_flasharraym_04.pdf`.

[18] SolidFire. SolidFire All-Flash Array. `http://info.solidfire.com/rs/solidfire/images/SolidFire_ProductDatasheet.pdf`.

[19] Nimble. Nimble All-Flash Array. `https://www.nimblestorage.com/technology-products/all-flash-array-specifications/`.

[20] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. SDF: Software-defined Flash for Web-scale Internet Storage Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, pages 471–484, 2014.

[21] Yongseok Oh, Eunjae Lee, Choulseung Hyun, Jongmoo Choi, Donghee Lee, and Sam H. Noh. Enabling Cost-Effective Flash based Caching with an Array of Commodity SSDs. In *Proceedings of the 16th Annual Middleware Conference*, Middleware'15, pages 63–74, 2015.

[22] Dimitris Skourtis, Dimitris Achlioptas, Noah Watkins, Carlos Maltzahn, and Scott Brandt. Flash on Rails: Consistent Flash Performance through Redundancy. In *Proceedings of the USENIX Annual Technical Conference*, ATC'14, pages 462–474, 2014.

[23] Myoungsoo Jung, Wonil Choi, John Shalf, and Mahmut Taylan Kandemir. Triple-A: A Non-SSD Based Autonomic All-Flash Array for High Performance Storage Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS'14, pages 441–454, 2014.

[24] Bo Mao, Hong Jiang, Suzhen Wu, Lei Tian, Dan Feng, Jianxi Chen, and Lingfang Zeng. HPDA: A Hybrid Parity-Based Disk Array for Enhanced Performance and Reliability. *ACM Transactions on Storage*, 8(1):26–52, 2012.

[25] Tzi cker Chiueh, Weafon Tsao, Hou-Chiang Sun, Tiang-Fang Chien, An-Nan Chang, and Cheng-Ding Chen. Software Orchestrated Flash Array. In *Proceedings of the 7th ACM International Systems and Storage Conference*, SYSTOR'14, pages 1–11, 2014.

[26] Feng Chen, David A. Koufaty, and Xiaodong Zhang. Understanding Intrinsic Characteristics and System Implications of Flash Memory Based Solid State Drives. In *Proceedings of the 11th ACM SIGMETRICS/International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'09, pages 181–192, 2009.

[27] Jaeho Kim, Donghee Lee, and Sam H. Noh. Towards SLO Complying SSDs Through OPS Isolation. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 183–189, 2015.

[28] Jens Axboe. FIO: Flexible I/O Tester. `https://github.com/axboe/fio`.

[29] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. SFS: Random Write Considered Harmful in Solid State Drives. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 12–12, 2012.

[30] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. F2FS: A New File System for Flash Storage. In *Proceedings of the 13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 273–286, 2015.

[31] Aayush Gupta, Youngjae Kim, and Bhuvan Urgaonkar. 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'09, pages 229–240, 2009.

[32] Guanying Wu and Xubin He. Reducing SSD Read Latency via NAND Flash Program and Erase Suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 10–10, 2012.

[33] Junghee Lee, Youngjae Kim, Galen M. Shipman, Sarp Oral, and Jongman Kim. Preemptible I/O Scheduling of Garbage Collection for Solid State Drives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(2):247–260, 2013.

[34] Ken Smith. Understanding SSD Overprovisioning. Flash Memory Summit 2012.

[35] Ji-Yong Shin, Mahesh Balakrishnan, Tudor Marian, and Hakim Weatherspoon. Gecko: Contention-Oblivious Disk Arrays for Cloud Storage. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies*, FAST'13, pages 285–297, 2013.

[36] Wiki. Logical Volume Manager. `https://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux)`.