MQSim-E: An Enterprise SSD Simulator

Dusol Lee, Duwon Hong, Wonil Choi, and Jihong Kim

Abstract—Compared to client SSDs, enterprise SSDs have rigorous performance standards such as high/consistent IOPS and low tail latencies. Unfortunately, we find that existing SSD simulation tools are not appropriate for a simulation of enterprise SSDs, due to lack of a few critical functionalities. We identify three such functionalities – (i) dynamic address allocation, (ii) fine-grained address mapping, and (iii) token-based garbage collection, and present an enterprise SSD simulator (called MQSim-E) by adding them into a widely-used (client) SSD simulator. Through extensive experiments, we demonstrate that our MQSim-E is a more suitable choice for emulating contemporary enterprise SSDs than existing state-of-the-art SSD simulators.

Index Terms—SSD simulators, enterprise SSDs, storage systems, performance evaluation

1 Introduction

During the past decade, solid-state drive (SSD) technologies have made great strides, thereby infiltrating everywhere from personal electronics to enterprise data centers. To facilitate SSDs to be efficiently integrated to these applications/systems, many SSD simulators have been proposed in the public domain [1], [2], [3], [4], [5]. Although these simulators are different in their modeling accuracy, supported functions and usage modes, most of existing SSD simulators are based on client SSDs.

Unlike client SSDs, enterprise SSDs need to satisfy much stringent I/O requirements of performance-driven applications (e.g., cloud/web services, big data analytics). Therefore, enterprise SSDs are required to support (i) high I/O operations per second (IOPS), (ii) high device endurance, and (iii) sustainable quality of service (QoS) by integrating more resources and/or implementing advanced functionalities within them. Since enterprise SSDs usually employ various *proprietary* optimization techniques to meet challenging I/O requirements, it is quite difficult for most SSD simulators to faithfully model these techniques. However, considering that challenging research topics in SSD storage systems are mostly from enterprise storage systems, the lack of suitable enterprise SSD simulators is a key obstacle to developing efficient enterprise SSDs with real-world impact.

In order to validate that existing SSD simulators are not suitable for a simulation of enterprise SSDs, we conducted an I/O performance experiment using two widely-used simulators, SimpleSSD [1] and MQSim [2], and two synthetic workloads. Both the workloads issue 4KB random requests; but, their read/write (R/W) ratios are different. Fig. 1 shows the IOPS of MQSim and SimpleSSD under varying size of I/O queue for the two workloads. To compare it with the IOPS of a real device, we refer to the datasheet of a modern enterprise SSD (Samsung PM9A3 [6]). Note that the architecture and timing configurations of the two simulators were configured to make their performance comparable to those of enterprise-class SSDs; more details can be found in Table 1. Surprisingly, we can observe that there is a huge gap in IOPS (at least by an order of magnitude) between the two simulation tools and the real device. Also, as can be seen in Fig. 1, the performance of the two simulators does not scale in an appreciable manner, while the IOPS of the real device increases (until it reaches a certain point) as the size of I/O queue increases.

According to our analysis, such low IOPS values of the two simulators originate from lack of a few critical functionalities therein. Even though they can configure physical specifications (e.g., the number/size of dies/blocks/pages, read/write latencies) of enterprise-scale devices, the absence of those functionalities may miss opportunities to boost up the performance or waste the device resources, which can prevent a simulated SSD from delivering the actual performance/endurance of contemporary enterprise SSDs.

Motivated by this, in this letter, we identify three such fundamental functionalities (§ 2) and develop an enterprise SSD simulator (called *MQSim-E*) by supporting the identified key functionalities. MQSim-E is built upon MQSim [2], since it models many critical features such as NVMe interface of modern enterprise SSDs. Using various workloads and real devices, we evaluate MQSim-E in terms of the performance, device endurance, and QoS sustainability (§ 3), and show that MQSim-E is a proper simulator for researching enterprise SSDs (MQSim-E is available at https://github.com/cares-davinci/MQSim-E.).

2 Functionalities of MQSim-E

2.1 Dynamic Address Allocation

In general, enterprise SSDs are designed to include as many parallel units as they can and take full advantage of the device-internal parallelism [7], [8]. However, we find that most of existing simulation tools fail to best utilize the parallel units (while they are able to configure them), and in turn, lead to quite low IOPS compared to real enterprise SSDs. More specifically, we observe that they do not exploit the *plane-level parallelism* effectively, while being good at leveraging other parallel units such as channels and dies. Fig. 8 shows the low plane utilization of them under various workloads.

Their low plane utilization stems from the *static address allocation* they employ. That is, they determine the physical address of a write I/O data (i.e., the location where the data is stored) based on its logical address using their own fixed rules [9]. Such a strategy may find quite limited opportunities to use the underlying planes in parallel, due to the constraints of using the plane-level parallelism.[1] Fig. 2a illustrates an example scenario where the physical addresses of eight write I/O requests are *statically* determined. Here, the two I/O requests whose logical addresses are `8` and `10` cannot be executed in parallel with the two I/O requests whose logical addresses are `0` and `2`; the same situation is for the four I/O requests whose logical addresses are `1`, `3`, `9`, and `11`. Consequently, four out of eight I/O requests stall, while there are many (four) planes that remain unused.

To address this, modern enterprise SSDs relax or remove these rules, and allow a write I/O data to be stored anywhere by allocating the corresponding physical address; this is called *dynamic address allocation*. This strategy can allocate the same physical address (but, of different planes) to arbitrary multiple write I/O

1. Two or more operations can be serviced by different planes in parallel, when they are of the same operation type (reads or writes) and have the same physical addresses across the different planes [10].

• *Dusol Lee, Duwon Hong, and Jihong Kim are with Seoul National University, Seoul 08826, South Korea. E-mail: {dslee, duwon.hong, jihong}@davinci.snu.ac.kr.*
• *Wonil Choi is with Hanyang University, Ansan 15588, South Korea. E-mail: wonilchoi@hanyang.ac.kr.*

Manuscript received 12 Dec. 2021; accepted 5 Jan. 2022. Date of publication 25 Jan. 2022; date of current version 31 Mar. 2022.
This work was supported in part by SNU-SK Hynix Solution Research Center (S3RC) and in part by the National Research Foundation of Korea (NRF) Grant funded by the Ministry of Science and ICT (MSIT) under Grant NRF-2021R1H1A209324011. The ICT at Seoul National University provided research facilities for this study.
(Corresponding author: Jihong Kim.)
Digital Object Identifier no. 10.1109/LCA.2022.3144773

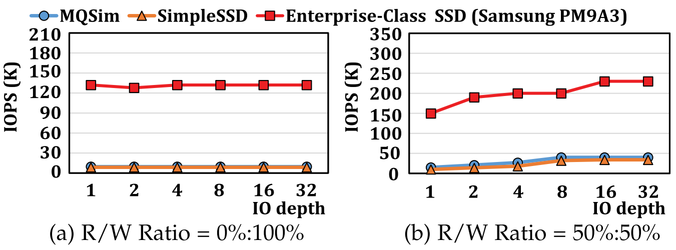


Fig. 1. The IOPS of two state-of-the-art SSD simulators (SimpleSSD [1] and MQSim [2]) under varying size of I/O queue for synthetic 4-KB random workloads with different Read/Write ratios. For the IOPS of an Enterprise-Class SSD, we refer to the datasheet of Samsung PM9A3.

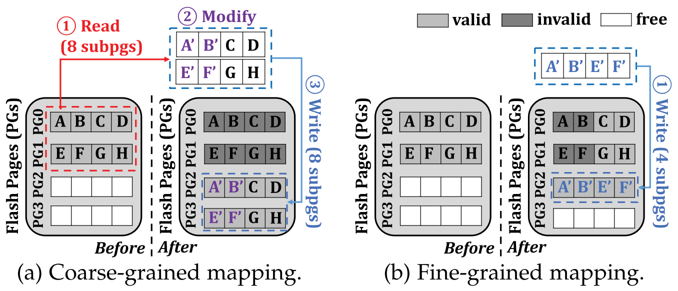


Fig. 3. The two different address mapping granularity and their impact on generating extra read and write operations.

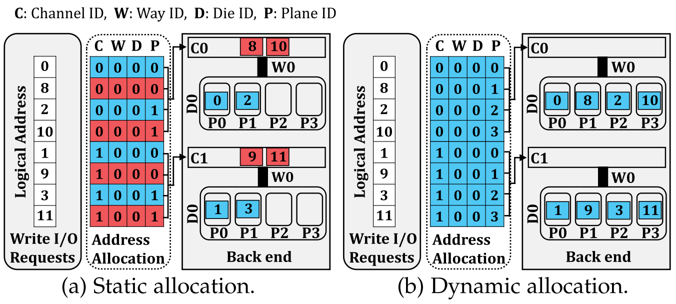


Fig. 2. The two different address allocation strategies and their impact on exploiting the plane-level parallelism.

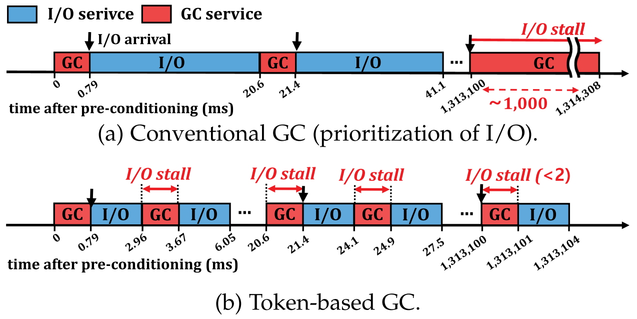


Fig. 4. The two different GC mechanisms and their impact on the consistency in the quality of I/O services. We used MQSim [2] as simulator and device configuration is same as Table 1. For simulations, synthetic 4-KB random workload with R/W Ratio = 0%:100% was used.

requests, and thus, create more opportunities to exploit the plane-level parallelism. Fig. 2b gives an example of the dynamic address allocation. Here, the physical addresses of all the eight I/O requests are determined in a way of leveraging all the planes simultaneously.

While there exist various different methods of dynamic address allocation [9], we implemented a *fully* dynamic allocation in our proposed simulator, since modern enterprise SSDs commonly employ such a policy for maximally exploiting the SSD-internal parallelism for write-intensive workloads[2] [7]. We also observed in our empirical study that our choice outperforms restricted dynamic allocation methods for tested enterprise workloads in terms of device performance and endurance.

2.2 Fine-Grained Address Mapping

It is common that contemporary enterprise SSDs experience a large number of small I/O requests; and, this scenario is becoming more prevalent, as vendors increase the size of flash page (e.g., up to 16KB) to increase the capacity and bandwidth of their products [11]. However, we observe that existing simulators handle such small I/O requests quite inefficiently. Specifically, they generate *redundant read and write operations* given a small write request, which leads to highly poor performance and endurance results, compared to those of real devices.

The inefficient small I/O handling of current simulators originates from their *large granularity of address mapping* – usually, the logical and physical addresses are mapped in the size of flash page. Under the page-granularity address mapping, a small write (update) I/O request is serviced by (i) reading the entire page data including the small data to be updated, (ii) modifying the corresponding portion of the page data, and (iii) writing back the entire page data; this is called *read-modify-write (RMW) operations* [11]. Fig. 3a depicts an example scenario where the page-granularity mapping generates RMW operations. Specifically, in order to service the four small writes `A'`, `B'`, `E'`, and `F'`, the two entire page data, `PG0` (including `A`, `B`, `C`, and `D`) and `PG1` (including `E`, `F`, `G`, and `H`), need to be read first. After the old data `A`, `B`, `E`, and `F` are

TABLE 1
The Compared Simulators, the Configurations of the Simulated Device, and the Tested Synthetic/Real Workloads

Compared simulators	`REF1 (SimpleSSD)`: SimpleSSD [1]. `REF2 (MQSim)`: MQSim [2].
	`OURS1 (MQSim+)`: MQSim [2] + § 2.1. `OURS2 (MQSim++)`: MQSim [2] + § 2.1 + § 2.2. `OURS3 (MQSim-E)`: MQSim [2] + § 2.1 + § 2.2 + § 2.3.
Reference device	`GOAL (Enterprise)`: Enterprise-Class SSDs (e.g., Samsung PM9A3 [6])
Device configurations	We collected the key parameters from various public sources, 1.92TB (OP:12.7%), 8 channels [6], 4 chips/channel [6], [18], 1 die/chip, 64GB die [18], 4 planes/die [18], 2,728 blocks/die, 16KB page[19], tR = 45us[20], tPROG = 650us[2] [20], tERASE= 3,500us [19]. Internal DRAM = 2GB (Mapping table: 2GB, Write buffer: 16MB)[2] . Device Queue size = 512[2] .
Synthetic workloads	(The ones used in the datasheet [6] of `GOAL (Enterprise)`). Request size: 4KB, 8KB, 16KB; Queue size: 1, 2, 4, 8, 16, 32. `W-EXTREME`: Read:Write = 0%:100%, `W-INTENSIVE`: Read:Write = 30%:70%, `Half-Half`: Read:Write = 50%:50%, `R-INTENSIVE`: Read:Write = 70%:30%, `R-EXTREME`: Read:Write = 100%:0%.
Real-world workloads	`varmail`, `fileserver`, `oltp` (filebench [17]). `prxy`, `prn`, `web` (MSR traces [16]).

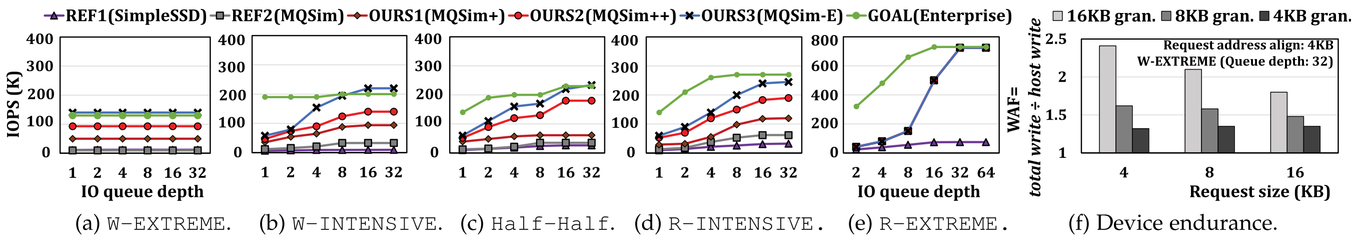


Fig. 5. (a)-(e): IOPS comparison of the tested simulators; (f) device endurance under varying granularity of address mapping.

modified to the new data A', B', E', and 'F', respectively, the two new page data (PG2 and PG3) are written. This process involves two page (eight subpages) read operations and two page (eight subpages) write operations.

To avoid these extra, heavy read and write operations, cutting-edge enterprise SSDs employ *fine-grained address mapping* – the logical and physical addresses are mapped in the sizes that are smaller than those of flash page. Doing so can service a small write right away by invalidating the corresponding old data, without reading and writing the rest (unmodified) part of the page data. Fig. 3b describes how the fine-grained address mapping streamlines the process of handling small I/O requests. The four small writes A', B', E', and F' can be written using a single page (four subpages) write operation, while the corresponding old data A, B, E, and F become invalid.

While it can reduce RMW operations for small requests, the fine-grained mapping increases the size of mapping table and the number of table accesses for large requests. However, such an overhead is negligible in a modern enterprise SSDs where internal large DRAM caches the entire mapping table[2] [2].

2.3 Token-Based Garbage Collection (GC) Management

One important condition of enterprise SSDs is their QoS sustainability (i.e., consistent IOPS, low tail latencies) [12]. According to our observation, however, the performance of existing SSD simulators is not consistent at all; that is, their IOPS fluctuates over time and their I/O response times are significantly increased. This is because the GC mechanisms they employ do not take the consistency in the quality of I/O services into account. As an example, Fig. 4a illustrates how the GC of MQSim [2] is executed. Specifically, immediate services of I/O requests are prioritized over GC operations, and thus, the GCs are usually postponed till they cannot be delayed any longer. Consequently, future I/O requests are likely to stall more frequently and for a longer time, due to the accumulated GCs.

To avoid this, modern enterprise SSDs employ GC mechanisms that can adjust the balance between I/O and GC. A popular choice is the *token-based GC management*[3], where token is a clean flash page to service a write I/O request. Specifically, a token is collected whenever GC reclaims a clean page, while each I/O can be serviced by discarding a token. This brings an effect of evenly distributing the GC overhead over many I/O requests, and in turn, rendering QoS across I/O requests uniform. Fig. 4b depicts a behavior of the token-based GC management and shows that I/O stall was reduced by almost 500 times compared to Conventional GC.

3 Evaluation and Verification

Methodology. Table 1 summarizes our experimental setups. We prepared three different versions of our simulator by adding each of the three functionalities one after the other: MQSim+ that adds only the dynamic address allocation (§ 2.1), MQSim++ that implements both the dynamic address allocations (§ 2.1) and the fine-grained address mapping (§ 2.2), and MQSim-E that includes all the three functionalities. We also employed SimpleSSD [1] (REF1(SimpleSSD)), MQSim [2] (REF2(MQSim)), and a reference device [6] (GOAL(Enterprise)). The architecture and timing configurations of the compared simulators are configured to make their performance comparable to enterprise-class SSDs; refer to key parameters in Table 1. On the SSD, we executed both the synthetic and real workloads [16], [17]. Especially, our synthetic workloads were generated to compare the performance results of the simulators with those of the real device. All the experimental data were collected after 95% of the total capacity of the SSD was filled with data, and thus, when the GC is frequently invoked.

IOPS. Figs. 5a, 5b, 5c, 5d, and 5e show the IOPS delivered by our tested simulators under varying depth of I/O queue, when each of the three synthetic workloads is executed. The closer IOPS to that of GOAL(Enterprise) a simulator achieves, the more accurately it can emulate the enterprise SSD. The IOPS of OURS3(MQSim-E) is quite close to those of the corresponding real device (GOAL(Enterprise)), which also scales as the size of I/O queue increases. Even OURS1(MQSim+) and OURS2(MQSim++) which does not employ the token-based GC mechanism but adopt the dynamic allocation and/or the fine-grained address mapping outperform REF1(SimpleSSD) and REF2(MQSim).

QoS Sustainability and Tail Latencies. Fig. 6 shows changes in the IOPS of the tested simulators over time, when W-EXTREME is executed. Overall, the IOPS of REF1(SimpleSSD) and REF2(MQSim) is not only quite low but also significantly fluctuating, due to GC invocations. In contrast, OURS3(MQSim-E) delivers consistent, high IOPS (that are close to that of the real device (GOAL(Enterprise)), regardless of GC invocations. Figs. 7a and 7b plot the CDF of I/O latencies of Half-Half at 75th and 99th percentiles. The token-based GC of OURS3(MQSim-E) is quite effective in the elimination of tail latencies.

Impact on Device Endurance. We use the WAF as a metric of device endurance (the lower the WAF is, the better the endurance is), and analyze the impact of address mapping granularity on the WAF values using OURS3(MQSim-E). Fig. 5f shows the WAF under varying granularity of address mapping (16KB, 8KB, 4KB) and size of I/O requests (4K, 8KB, 16KB). When the mapping granularity is large (e.g., 16KB), the WAF increases significantly as the size of I/O requests decreases. In contrast, when the fine-grained mapping (e.g., 4KB) is supported, one can expect low WAF values, regardless of the size of requests.

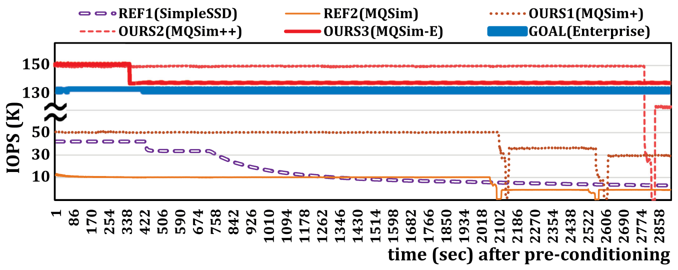
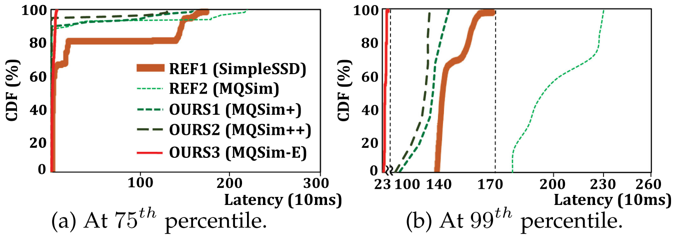


Fig. 6. The changes of IOPS over time in W-EXTREME.

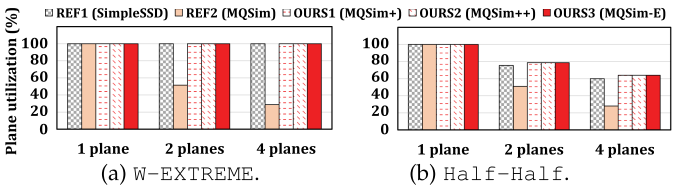
2. Private communication with commercial SSD firmware developers.

3. For example, [13], [14], [15] are based on a token-based GC management idea, although there are differences in underlying assumptions and management policies. In our proposed simulator, we adopted a simplified version of more advanced token-based schemes.



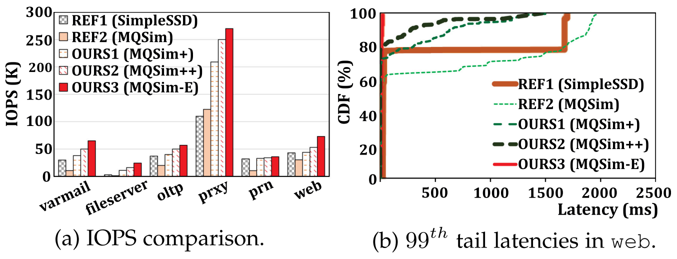
(a) At 75th percentile. (b) At 99th percentile.

Fig. 7. The CDF of read I/O latencies in `Half-Half`.



(a) `W-EXTREME`. (b) `Half-Half`.

Fig. 8. Plane utilization under varying number of planes.



(a) IOPS comparison. (b) 99th tail latencies in `web`.

Fig. 9. Results of real-world workloads.

TABLE 2
The Existence of the Three Functionalities

Existing SSD Simulators	Dynamic Addr. Allocation (§ 2.1)	Fine-grained Addr. Map (§ 2.2)	Token-based GC Mgmt (§ 2.3)
DiskSim [3]	o	x	x
FlashSim [4]	x	x	x
FEMU [5]	o	x	x
SimpleSSD [1]	o	x	x
MQSim [2]	x	x	x
MQSim-E (ours)	o	o	o

Analysis on Plane Utilization. Increased plane-level parallelism is one major contributor to the performance improvement of our simulators; recall the dynamic address allocation (§ 2.1). Fig. 8 shows the plane utilization when the number of planes in a die varies from one to four. Specifically, our simulators find more or at least comparable chances to use multiple planes in parallel, campared to `REF1(SimpleSSD)` and `REF2(MQSim)`. According to our analysis, `REF2(MQSim)` uses only one plane in most of the execution time due to its static address allocation.

Results of Real-World Workloads. Fig. 9a compares the IOPS of all the tested simulators for the six real workloads that exhibit various I/O characteristics/patterns; note that results of these workloads on `GOAL(Enterprise)` are not present in the datasheet. One can observe that `OURS3(MQSim-E)` still delivers high IOPS and consistent I/O latencies for real-world workloads, against `REF1(SimpleSSD)` and `REF2(MQSim)`. Fig. 9b plots the CDF of read I/O latencies (at 99th percentile) in a workload (`web`). `OURS3(MQSim-E)` eliminates most of the tail latencies, while others experience a large amount of long I/O latencies.

4 Related Work

There are many publicly available SSD simulators [1], [2], [3], [4], [5]. Even though existing tools have been gradually improved one after the other for a more detailed, accurate SSD simulation, *none of them are appropriate for a realistic simulation of enterprise SSDs, due to the lack of the critical functionalities*. Table 2 shows whether each of the simulators implement the three critical functionalities or not.

5 Conclusion

We presented MQSim-E, an enterprise SSD simulator, that can accurately mimic the behaviors of modern enterprise SSDs. MQSim-E focused on supporting three performance-enhancing techniques by maximizing the device-internal parallelism, minimizing redundant data movements, and adaptively balancing workloads from GC and host I/O. Using the synthetic and real workloads, we validated that the performance/behavior of MQSim-E is quite close to a real enterprise SSD.

References

[1] D. Gouk *et al.*, "Amber*: Enabling precise full-system simulation with detailed modeling of all SSD resources," in *Proc. 51st Annu. IEEE/ACM Int. Symp. Microarchit.*, 2018, pp. 469–481.
[2] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, O. Mutlu, "MQsim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. 16th USENIX Conf. File Storage Technol.*, 2018, pp. 49–66.
[3] N. Agrawal *et al.*, "Design Tradeoffs for SSD Performance," in *Proc. USENIX Annu. Tech. Conf.*, 2008, pp. 57–70.
[4] Y. Kim, B. Tauras, A. Gupta and B. Urgaonkar, "FlashSim: A simulator for nand flash-based solid-state drives," in *Proc. 1st Int. Conf. Adv. Syst. Simul.*, 2009, pp. 125–131.
[5] H. Li, M. Hao, M. H. Tong, S. Sundararaman, M. Bjørling, H. S. Gunawi, "The CASE of FEMU: Cheap, accurate, scalable and extensible flash emulator," in *Proc. 16th USENIX Conf. File Storage Technol.*, 2018, pp. 83–90.
[6] Samsung, *PM9A3 SSD Whitepaper*, 2021. [Online]. Available: https://semiconductor.samsung.com/ssd/datacenter-ssd/pm9a3/
[7] D. Hong *et al.*, "Reparo: A Fast RAID Recovery Scheme for Ultra-large SSDs," *Trans. Storage*, vol. 17, no. 3, pp. 1–24, 2021.
[8] I. Micron Technology, *NAND Flash Media Manage. Through RAIN*, 2011. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/technical-marketing-brief/brief_ssd_rain.pdf
[9] T. Arash *et al.*, "Performance evaluation of dynamic page allocation strategies in SSDs," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 2, pp. 1–33, 2016.
[10] C. Gao, L. Shi, C. J. Xue, C. Ji, J. Yang and Y. Zhang, "Parallel all the time: Plane level parallelism exploration for high performance SSDs," in *Proc. 35th Symp. Mass Storage Syst. Technol.*, 2019, pp. 172–184.
[11] M. Kang, W. Lee, S. Kim, "Subpage-based flash translation layer for solid state drivers," in *Proc. Korea Adv. Inst. Sci. Technol. Open Access Self Arch. Syst.*, 2016, pp. 1–30.
[12] Kingston., The Right Solid-State Drive (SSD) Matters, 2020. [Online]. Available: https://www.kingston.com/czech/en/solutions/servers-data-centers/ssd-matters
[13] S. Choudhuri, T. D. Givargis, "Deterministic service guarantees for NAND flash using partial block cleaning," in *Proc. 6th IEEE/ACM/IFIP Int. Conf. Hardware/Softw. Codesign Syst. Synth.*, 2008, pp. 19–24.
[14] M. Jung, W. Choi, S. Srikantaiah, J. Yoo and M. T. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," in *Proc. ACM/IEEE 41st Int. Symp. Comput. Archit.*, 2014, pp. 289–300.
[15] Z. Sha *et al.*, "Low I/O intensity-aware partial GC scheduling to reduce long-tail latency in SSDs," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–25, 2021.
[16] A. Rowstron, A. Donnelly, D. Narayanan, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, pp. 1–23, 2008.
[17] V. Tarasov *et al.*, "Filebench: A flexible framework for file system benchmarking," *USENIX; Login*, vol. 41, no. 1, 2016, pp. 6–12.
[18] ANANDTECH, "2021 NAND Flash Updates from ISSCC: The Leaning Towers of TLC and QLC," 2021. [Online]. Available: https://www.anandtech.com/show/16491/flash-memory-at-isscc-2021
[19] D. Kang *et al.*, "13.4 A 512Gb 3-bit/Cell 3D 6th-Generation V-NAND Flash Memory with 82MB/s Write Throughput and 1.2Gb/s Interface," in *Proc. IEEE Int. Solid- State Circuits Conf.*, 2019, pp. 216–218.
[20] Samsung, "Samsung electronics takes 3D memory to new heights with sixth-generation V-NAND SSDs for client computing," Samsung Newsroom, 2019. [Online]. Available: https://semiconductor.samsung.com/newsroom/news/samsung-electronics-takes-3d-memory-to-new-heights-with-sixth-generation-v-nand-ssds-for-client-computing/