Improving Overall Performance of TLC SSD by Exploiting Dissimilarity of Flash Pages

Wenhui Zhang, Qiang Cao, *Senior Member, IEEE*, Hong Jiang, *Fellow, IEEE*, and Jie Yao, *Member, IEEE*

Abstract—TLC flash has three types of pages to accommodate the three bits in each TLC physical cell exhibiting very different program latencies. This paper proposes PA-SSD to effectively improve the overall performance by exploiting the dissimilarity of TLC pages on program latency throughout the write request handling workflow. The main idea behind PA-SSD is to coordinately allocate the same type of pages for sub-requests of any given user write request, to mitigate the potential program latency imbalance among the sub-requests, and to schedule sub-requests according to their page-types. We achieve the PA-SSD design goal by answering three key research questions: (1) how to properly determine page-type for each user write request? (2) how to actually allocate a physical page for each sub-request with an assigned page-type from (1)? (3) how to effectively schedule the sub-requests in the chips queues when their page-types are judiciously allocated from (2)? To answer the first question, we propose seven page-type specifying schemes to investigate their effects under different workloads. We answer the second question by redesigning the page allocation strategy in TLC SSD to uniformly and sequentially determine physical pages for allocation following the internal programming process of TLC flash. Lastly, a page-type aware scheduling policy is presented to reorder the sub-requests within chips' queues. Our experiments show that PA-SSD can accelerate both the write and read performance. Particularly, our proposed queue-depth based page-type specifying scheme improves write performance by 2.6 times and read performance by 1.5 times over the conventional TLC SSD.

Index Terms—TLC SSD, diverse program latencies, TSU scheduling, write performance, page-type aware

1 Introduction

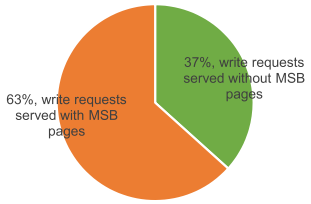
TLC (Triple-Level Cell) flash is gradually becoming a dominant storage media in Solid-State Drives (SSDs) because of its higher storage capacity and lower price per gigabyte than SLC (Single-Level Cell) flash and MLC (Multi-Level Cell) flash. However, TLC, which stores three data bits in each physical cell, requires finer-grained program steps, resulting in higher program latency than SLC and MLC flash [1], [2]. This increased program latency leads researchers and developers to propose new SSD designs to boost the TLC SSD performance.

Because the three bits in a TLC cell, LSB (Least Significant Bit), CSB (Central Significant Bit) and MSB (Most Significant Bit), exhibit very different program latencies, the TLC SSD separates these bits into three types of pages with diverse program latencies, i.e., LSB pages, CSB pages, and MSB pages [1], [3]. Specifically, an LSB page has the shortest program latency (e.g., 500 μs), a CSB page has the medium program latency (e.g., 2000 μs), and an MSB page has the longest program latency (e.g., 5500 μs) [3]. Due to the high program latencies of CSB and MSB pages, write requests served with CSB and MSB pages usually have much longer response times than with LSB pages (up to 10x). To boost TLC write performance, many proposals suggest enabling the SLC mode in which only LSB pages are used when serving user write requests [1], [4], [5], [6], [7]. However, these methods waste some of the storage capacity provided by the CSB and MSB pages that account for up to 2/3 of the total capacity. On the other hand, the conventional SSD design allocates pages for user write requests without differentiating page-types. As each user write request greater than a page in size is partitioned into multiple page-sized sub-requests that are then allocated pages independently of page-types [8], a large proportion of user write requests are actually served with at least one MSB page in a conventional TLC SSD, as shown in Fig. 1a, resulting in considerable write inefficiency. This motivates us to redesign the page allocation strategy so that it takes page-type into consideration to improve write performance of TLC SSD without sacrificing any storage capacity.

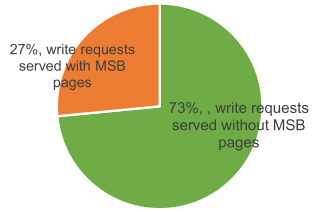
In this paper, we present a **P**age-type **A**ware design of TLC SSD, or PA-SSD, to exploit the diverse program latency of TLC pages throughout the write request execution workflow. Specifically, it coordinately allocates pages of the same type for the sub-requests of a given user write request, so as to significantly lower the percentage of write requests served with MSB pages, as shown by Fig. 1b, and thus improve the

- *W. Zhang and Q. Cao are with the Wuhan National Laboratory for Optoeletronics, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: Singularity_x@outlook.com, caoqiang@hust.edu.cn.*
- *H. Jiang is with the Computer Science and Engineering Department, University of Texas at Arlington, Arlington, TX 76019. E-mail: hong.jiang@uta.edu.*
- *J. Yao is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: jackyao@hust.edu.cn.*

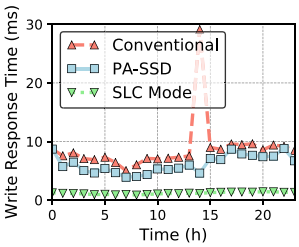
Manuscript received 8 Oct. 2018; revised 5 Aug. 2019; accepted 7 Aug. 2019. Date of publication 14 Aug. 2019; date of current version 26 Dec. 2019. (Corresponding author: Wenhui Zhang.) Recommended for acceptance by M. D. Santambrogio. Digital Object Identifier no. 10.1109/TPDS.2019.2934458



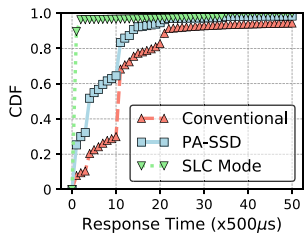
(a) A majority of user write requests are served with MSB pages in conventional SSD design.



(b) The proportion of user write requests served with MSB pages is reduced in PA-SSD design.



(c) The write performance over time.



(d) The cumulative distribution function of write response time.

Fig. 1. Simulation results on tDAP workload. The "SLC Mode" corresponds to TLC SSD that all flashes are used in SLC mode.

write performance. In other words, PA-SSD always attempts to use the same type of pages to serve all sub-requests of any single write request. To make this possible, three challenging questions must be answered: Q1. How to properly determine page-types for user write requests in runtime? Q2. How to actually allocate pages to sub-requests of a user write request whose page-type has been specified (by an answer to Q1)? Q3. How to schedule the sub-requests in the chips queue when their page-types are judiciously allocated (by an answer to Q2)?

For the first challenge, we consider various factors, such as host requirements, request sizes, device-level queue length, and so on, and accordingly propose seven page-type specifying schemes. They are *sU* (Uniformly specification) scheme, *sHG* (Host-Guided specification) scheme, *sLF* (LSB-First specification) scheme, *sSB* (Size-Based specification) scheme, *sQD* (Queue-Depth-based specification) scheme, *sWB* (Write-Buffer-based specification) scheme, and *sUB* (Utilization-Based specification) scheme. We also evaluate these schemes (and some combinations) and reveal that a combination of *sQD* and *sUB* behaves the best on providing high and stable write performance in PA-SSD.

For the second challenge, while the conventional TLC SSD allows a single candidate page and a single active block within a plane (i.e., no choice), PA-SSD provides more than one candidate page and three active blocks within each plane, and uses a redesigned page allocation strategy to select suitable candidate pages for write sub-requests according to their assigned page-types.

For the third challenge, PA-SSD employs a page-type aware sub-request scheduling, namely PAS, in legacy Transaction Scheduling Unit (TSU), for further improving write performance. Specifically, PAS strategically promotes sub-requests allocated with LSB pages in the chip queue. The insight behind this policy is that sub-requests with LSB can be fast accessed, thus reducing the average waiting time of the queue.

By integrating these techniques, PA-SSD outperforms conventional SSD significantly. As shown in Fig. 1b (with more details in Section 6), the proportion of user write requests served with at least one MSB page in PA-SSD is much lower than that of the conventional TLC SSD (Fig. 1a), leading to lower write response time, as illustrated in Figs. 1c and 1d.

In summary, in proposing and studying PA-SSD, we aim to make the following contributions in this paper:

1) We analyze the drawbacks of type-blind page allocation strategy of the conventional TLC SSD that allocates pages for the sub-requests of a user write request regardless of page-types.
2) We present PA-SSD, a page-type aware TLC SSD design that first determines a proper page-type for serving a given user write request and then coordinately allocates pages of the required type for all sub-requests of the request. Seven schemes are designed in PA-SSD to determine page-types for user write requests, while the page allocation strategy in the conventional TLC SSD is redesigned by appropriately relaxing the program constraints within planes to realize the coordinated, type-specified page allocation to sub-requests of any write request.
3) We propose Page-type Aware Sub-request (PAS) scheduling policy in the TSU. It reorders write sub-requests according to their assigned page-types, ensuring LSB pages to be prioritized.
4) We simulate PA-SSD with SSDSim and evaluate its performance in terms of write/read response times on eight typical real-world workloads. Our experimental results show that PA-SSD significantly improves both the write and read performances of the conventional TLC SSD without any sacrifice to storage capacity and P/E cycle endurance. Especially, by using the combination of the *sQD* and *sUB* page-type specifying schemes, PA-SSD improves the write and read performances of the conventional TLC SSD by 2.6x and 1.5x on average, respectively.

The remainder of this paper is organized as follows. In Section 2, we present background of TLC SSD. Section 3 motivates the PA-SSD proposal with insightful observations and analysis. The detail design of PA-SSD is presented in Section 4. In Sections 5 and 6, we present our experimental setups and results for demonstrating the efficacy of PA-SSD. Section 7 describes related works on improving write performance of TLC SSD. Finally, we conclude this paper in Section 8.

2 Background

2.1 SSD Architecture

As shown in Fig. 2, an SSD is composed of three primary components, i.e., host interface, SSD controller, and flash chip array [8]. The host interface supports communication between the host system and SSD controller, and maintains the device-level I/O queue [9]. The SSD controller, usually containing an embedded processor and DRAM, is responsible for handling I/O requests and managing SSD resources by executing a set of flash translation layer (FTL) functions,

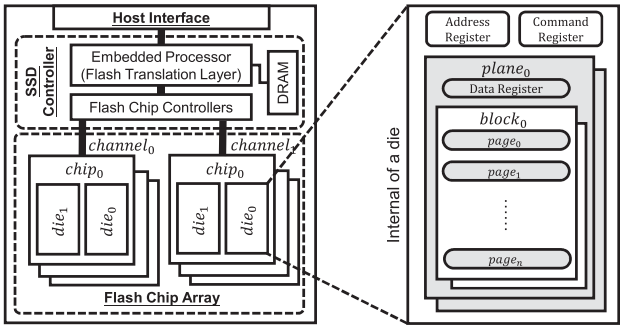


Fig. 2. Architecture of SSD.

e.g., address translation, garbage collection, and wear leveling. The SSD controller also communicates with the flash chip array through the flash chip controllers. The flash chip array composed of multiple flash chips is connected to the flash controller via channels and provides the physical storage capacity. Flash chips are composed of dies, each of which has its own address register and command register. Dies are further divided into planes. Within a plane, pages, the atomic units for read and program (write) operations, are grouped into blocks to form the atomic units for the erase operation. Importantly, page read and program operations can be striped across channels, chips, dies, and planes for parallel processing [8], [9], [10].

2.2 Write Request Execution Workflow

In Fig. 3, we illustrate the write request execution workflow within a conventional SSD. Upon the arrival of a write request from the host at the host interface of SSD, the latter first queues the request in the device-level I/O queue (I/O queue for short) and then partitions the request into page-sized sub-requests,[1] each with a specific LPA (Logical Page Address) [7]. These sub-requests are then sent to the SSD controller for address translation, which is an important function of FTL that translates the LPA to PPA (Physical Page Address). The address translation for write sub-requests is also referred to as *Page Allocation*. The page allocation selects free pages for sub-requests via two primitives, the *PLAlloc* primitive that allocates channel ID, chip ID, die ID, and plane ID, and the *BLAlloc* primitive that allocates block ID and page ID [8]. Finally, a PPA is determined by the combination of these six IDs, and the mapping pair (LPA, PPA) is stored into the page-level mapping table for future read operations. With page allocation accomplished, the sub-requests are inserted into corresponding chips' queue, where not only write sub-requests (program operation) but also read sub-requests (read operation) and erase operations are scheduled by the Transaction Scheduling Unit (TSU) for purpose of improving parallelism [9], [11], reducing response time [12], [13], providing fairness [14], etc. Finally, the sub-requests are delivered to flash controllers where they are striped across channels/chips/dies/planes for parallel programming [8]. When handling a program operation, the flash controller transfers the command and address information to the target die and the user data to the target plane. The user data is cached in the data register of the target plane before being programmed to the target page. In this study, a page-sized sub-request is considered finished when its corresponding user data is physically programmed,[2] and a user write request is considered completed when all of its sub-requests are finished.

2.3 Page-Types and their Diverse Program Latencies

TLC flash stores three bits with different program latencies, namely, LSB (Least Significant Bit), CSB (Central Significant Bit), and MSB (Most Significant Bit) within each flash cell. The bits of the same type (program latency) in cells of a wordline form a page. Therefore, pages in TLC flash are of three different types of LSB, CSB, and MSB. Conventionally, the three differently typed pages within a wordline are programmed separately page-by-page [1], [15], and pages can be read before the wordline is fully-programmed (i.e., all three pages are programmed). Many existing studies revealed that the three types of pages have significantly diverse program latencies [1], [3]. Typically, for the 25 nm TLC flash, LSB, CSB, and MSB pages exhibit 500 μs, 2000 μs, and 5500 μs program latencies, respectively [3]. In addition, programming a(n) CSB(MSB) page requires that the associated LSB(LSB and CSB) page(s) be accessed first, resulting in even longer program latency. To mitigate the performance impact of this requirement of physically reading extra pages when programming CSB and MSB pages, the LSB and CSB pages within an un-fully-programmed wordline are usually buffered in DRAM in conventional TLC SSD.

2.4 Block Management and Page Management

Within a plane, blocks are partitioned into two pools, i.e., *free pool* and *used pool*, and one *active block*, as depicted in Fig. 4. Specifically, erased blocks (all pages within are free) are belong to the *free pool*, while fully-programmed blocks (all pages within are programmed) are belong to the *used pool*. In addition, each plane maintains an *active block*, which usually contains programmed pages as well as free pages, for serving subsequent page-sized write sub-requests. When all the pages within the *active block* are programmed, the block is deactivated and inserted into the *used pool*, while another block is obtained from the *free pool* and activated as a new active block. On the other hand, once the *free pool* size is beneath a GC (garbage collection) threshold, GC operation is triggered and it reclaims blocks from *used pool* to *free pool*.

Within the active block, pages should be programmed sequentially according to their IDs [16], as illustrated by the example depicted in Fig. 5, for the purpose of reducing cell-to-cell program interference to fully-programmed wordlines.

According to this block management within plane and page management within block, during page allocation, after the target plane is determined by the *PLAlloc* primitive, there is actually only one candidate page — the only next free page of the active block can be allocated by the *BLAlloc*

1. A sub-request referred in this article also is called a transaction in some other literatures. We use the term sub-request because it indicates the set membership, i.e., a request contains a set of sub-requests, more clearly.

2. In some studies, a sub-request is regarded finished when its corresponding data is cached in the data register of the target plane, resulting in lower response time. However, the data actually is not permanently stored until it is programmed.

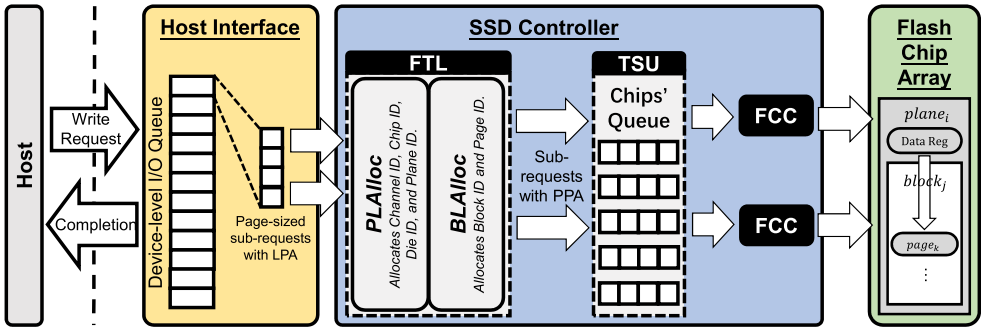


Fig. 3. Write request execution workflow in a conventional SSD. The FTL, TSU, and FCC are short for Flash Translation Layer, Transaction Scheduling Unit, and Flash Chip Controller, respectively.

primitive as the target page. This design of a lone candidate page per plane can significantly simplify the management of blocks and pages, however, at the expense of reduced flexibility of allocating a desired type of page.

2.5 TSU Scheduling

Transaction Scheduling Unit, TSU for short, reorders sub-requests in chips' queue, including write, read, and erase operations. There are two basic TSU policies as the first-come-first-serve (FCFS) and read-priority (RP) [14]. For FCFS, read and write sub-requests have the same priority while erase operations usually have the lowest priority except that a GC (garbage collection) hard threshold is exceeded, where the priority of erase operations are shortly promoted to the highest. Different from FCFS, the RP policy gives read sub-requests higher priority than write sub-requests for ensuring low read response time. Other than these two basic policies, researchers also presented several advanced policies for reducing response time (e.g., Slacker [13]) and providing fairness between different I/O streams (e.g., FLIN [14]).

3 Motivation

In TLC SSD, a user write request greater than a page in size is first partitioned into multiple page-sized sub-requests that are then allocated pages by the page allocation strategy and striped across channels/chips/dies/planes for parallel processing [8]. A user write request completes when all of its constituent sub-requests are finished. Thus, the response time of a user write request actually is determined by its slowest sub-request. As the program latency of MSB page is far longer than those of LSB page and CSB page in TLC flash, the response time of a user write request with at least one MSB page involved will be much longer than one without, particularly one with all its sub-requests allocated LSB pages. We categorize user write requests in TLC SSD into three distinctive groups, i.e., *fast writes* (all sub-requests are served by LSB page), *medium writes* (sub-requests are served by at least one CSB page but no MSB pages), and *slow writes* (at least one sub-request is served by MSB page), in order of increasing process time. Unfortunately, a large proportion of user write requests actually are *slow writes* in conventional TLC SSD, as illustrated in Fig. 1a, leading to considerable write inefficiency of TLC SSD.

The main reason for the MSB write dominance is that, in conventional SSD design, the page allocation strategy allocates pages for the sub-requests of a user write request independently of the page-type. In other words, each sub-request has a 1/3 probability of being served by an MSB page. Accordingly, for a user write request with n sub-requests involved, the probability of at least one of its sub-requests being served by an MSB page (*slow write*), denoted as P_{slow}, is equal to $(1-(\frac{2}{3})^n)$. On the other side, the probability of all sub-requests being served by LSB pages (*fast write*), denoted as P_{fast}, is equal to $(\frac{1}{3})^n$. In addition to these two cases, the sub-requests also may be served by at least one CSB page but no MSB pages (*medium write*), and the probability of this happening, denoted as P_{medium}, is equal to $((\frac{2}{3})^n-(\frac{1}{3})^n)$. For user write requests with only one sub-request involved ($n=1$), the three probabilities are the same and equal to 1/3. However, P_{slow} increases with n very quickly while the other two probabilities decrease with n, as illustrated in Fig. 6. For instance, when $n=4$, $P_{slow}=80\%$, meaning that most of the user write requests are *slow writes*. On the contrary, at $n=4$, P_{fast} and P_{medium} decrease to 1 and 19 percent, respectively, diminishing the desirable impacts of *fast/medium writes*.

This lopsided negative performance impact of MSB write dominance in conventional TLC SSD, where the undesirable *slow write* increases rapidly while the desirable *fast write* decreases rapidly with the request size n (the number of

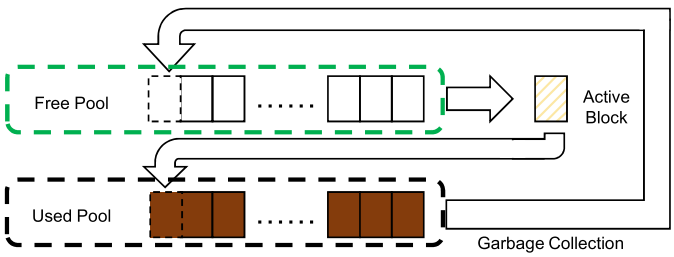


Fig. 4. Block management in conventional SSD. Blocks are partitioned into two pools as *free pool* and *used pool*. One *active block* is provided for serving write sub-requests.

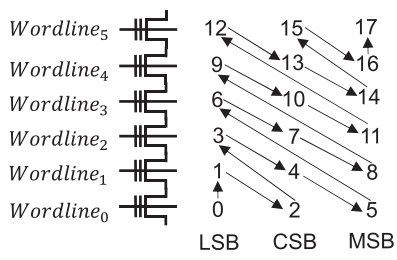


Fig. 5. Program order of pages within a TLC block [3], [16].

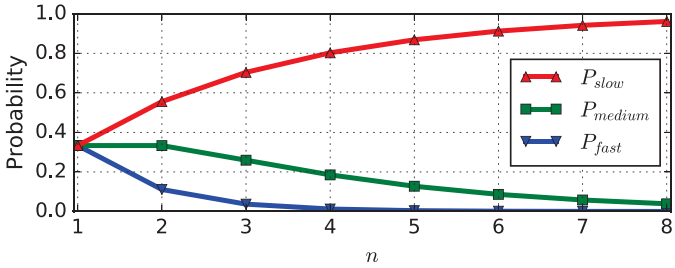


Fig. 6. The probabilities of *fast write* (P_{fast}), *medium write* (P_{medium}), and *slow write* (P_{slow}) as a function of n (the number of sub-requests).

sub-requests involved), motivates us to design a new page allocation strategy that minimizes *slow writes* while maximizes *fast writes*. In other words, the new strategy should keep P_{slow} low and P_{fast} high as n increases. To this end, we suggest a coordinative page allocation scheme that *allocates the same type of pages for the sub-requests of a given user write request*. Furthermore, we propose to proactively and judiciously specify the type of page used to serve each user write request. In so doing, we hope to be able to control the P_{slow}, P_{medium}, and P_{fast} values to improve the write performance of TLC SSD. Therefore, we present our novel page-type aware TLC SSD design, PA-SSD, to be elaborated next.

4 Page-type Aware TLC SSD

4.1 Overview

The proposed **PA**-SSD is a **P**age-type **A**ware TLC SSD design that improves the overall performance by fully exploiting the dissimilarity of flash pages throughout the write requests execution workflow. Especially, PA-SSD attempts to use the same type of pages for serving sub-requests of any single user write request. To this end, PA-SSD should first determine which type of page is used for serving a write request, then allocates pages for corresponding sub-requests according to the determined page-type, and finally reorder the sub-requests in the chips queue.

In Fig. 7, we illustrate the write request execution workflow in PA-SSD. There are three major differences between the write request execution workflow in PA-SSD design and that in conventional SSD design (illustrated in Fig. 3). First, the host interface in PA-SSD proactively assigns a type of page for each user write request according to specific *page-type specifying schemes*, detailed in Section 4.2. Second, PA-SSD allocates block IDs and page IDs for the page-sized sub-requests according to their assigned page-types with a redesigned *pa-BLAlloc* primitive, detailed in Section 4.3. These two modifications together make *page-type aware page allocation* possible. Third, a *page-type aware sub-request scheduling policy* is designed and introduced into the transaction scheduling unit of PA-SSD for further shortening the write response time, elaborated in Section 4.4.

4.2 Page-Type Specifying Schemes

The host interface in PA-SSD is responsible for determining and assigning the page-type for each user write request. Note that assigning page-type for a user write request actually only entails adding some attributes to its sub-requests, informing the page allocation strategy which type of pages should be allocated to these sub-requests. In the next subsection, we detail the type-specified page allocation strategy in PA-SSD that is responsible for actually allocating pages for sub-requests according to their assigned page-types.

A user write request assigned with the LSB(CSB/MSB) page is expected to be a *fast(medium/slow) write*. Thus, proactively determining the page-types for user write requests has the potential to adjust the ratios of *fast/medium/slow writes* to optimize the write performance of TLC SSD. To accommodate various performance requirements, we propose the following seven schemes in PA-SSD to determine the page-types for user write requests.

Uniformly Specification (sU). With this scheme, PA-SSD assigns the three page-types, i.e., LSB, CSB, and MSB, for user write requests randomly or in a round-robin style. Therefore, a write request will be assigned any of these three page-types equally likely with a probability of 1/3, leading to a uniform distribution of *fast/medium/slow writes*.

Host-Guided Specification (sHG). For a write command in the NVMe policy, the host can specify the requirement of response time by setting the two-bits long attribute 'Access Latency' [17]. Accordingly, with the sHG scheme, PA-SSD assigns LSB, CSB, and MSB pages for write requests requiring short, medium, and long response times, respectively. When the 'Access Latency' of a write request is omitted by the host, PA-SSD will assign its page-type according to other

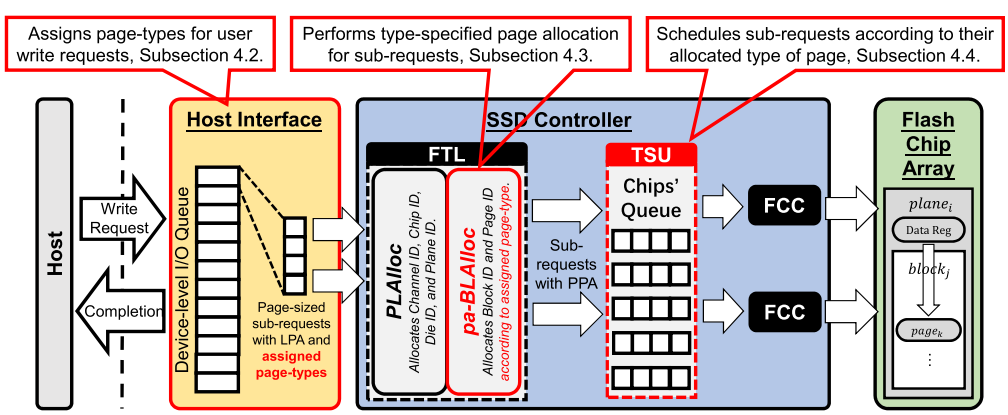


Fig. 7. Write request execution workflow in PA-SSD, which has three major differences compared with that in conventional SSD. First, for each user write request, the host interface of PA-SSD proactively assigns a page-type that is inherited by all of its constituent page-sized sub-requests. Second, the *BLAlloc* primitive is replaced by *pa-BLAlloc* that allocates pages for sub-requests according to their assigned page-types. Third, a novel scheduling policy is introduced into the TSU of PA-SSD to reorder sub-requests according to their allocated page-types.

schemes, such as the sU scheme. By employing sHG, PA-SSD effectively provides an interface for the host to leverage the diverse program latencies within TLC flash, improving the QoS (quality of service) of different applications.

LSB-First Specification (sLF). With sLF, PA-SSD always assigns the LSB pages to any user write requests. The insight behind this scheme is to use the LSB pages to maximally and greedily improve the write performance [3], [18]. However, when this scheme is employed, LSB pages tend to be used up very quickly, leaving only CSB pages and MSB pages to serve subsequent write requests (with more details in Section 6.3). Thus, sLF is suggested as a turbo mode to improve write performance during write-intensive bursts.

Size-Based Specification (sSB). In most cases, small-sized user write requests are expected to have shorter response times while large-sized requests are less sensitive to response time [19]. Accordingly, with the sSB scheme, PA-SSD assigns LSB pages for small-sized requests (e.g., requests smaller than 8 KB). On the other hand, for large-sized requests (e.g., requests larger than 8 KB), PA-SSD determines their page-type by other schemes, e.g., sU.

Queue-Depth-based Specification (sQD). The I/O intensity of a SSD can be sensed by the length of the device-level I/O queue, which is maintained in the host interface. During busy times when the queue is long, the response times of requests (both read and write requests) are dominated by their waiting time. By shortening the program latency of each user write request, the waiting time can be greatly reduced, resulting in high write and read performance. Accordingly, PA-SSD with the sQD scheme assigns LSB pages for all write requests when the device-level I/O queue is longer than a preset threshold, e.g., 10 requests. On the other hand, when the queue is shorter than the threshold, another scheme is employed for assigning page-types, e.g., sU and sUB schemes. The sQD scheme is regarded as a dynamic-sLF scheme that activates and deactivates sLF dynamically according to the I/O queue depth. The queue depth threshold of sQD determines how aggressive of sQD to take use of LSB pages. In fact, sQD with the queue depth threshold of 0 is reduced to sLF.

Write-Buffer-based Specification (sWB). In [18], Park et al. presented the idea of proactively allocating LSB pages for sub-requests according to the utilization of write buffer (DRAM) within MLC SSD. It allocates LSB pages for sub-requests when the write buffer is full. This idea can also be applied to determining page-types for user write requests in PA-SSD, resulting in the sWB scheme that assigns the LSB pages for user write requests when the write buffer is full. Compared with the sQD scheme that senses both write and read intensity from the device-level I/O queue depth, the sWB scheme can only sense write intensity, missing the opportunity to improve read performance by speeding up write requests during read intensive periods. In addition, the write buffer may be filled by latency-insensitive large-sized writes, leading to a waste of LSB pages when sWB is employed. In fact, when the write buffer is very small, the sWB scheme actually becomes the sLF scheme.

Utilization-Based Specification (sUB). The extremely imbalanced use of the three types of pages can result in inefficient garbage collection because some blocks may be reclaimed before all pages are programmed (e.g., all MSB pages are not programmed). By assigning page-types for requests according to the respective free capacities of the three page-types, sUB-based PA-SSD can effectively balance their utilizations. With sUB, PA-SSD determines page-types for write requests according to three probabilities P_L, P_C, and P_M, namely, those of assigning a write request with LSB, CSB, and MSB page-types respectively. To accommodate the utilization of the three page-types, the sUB scheme sets $P_L : P_C : P_M = \#LSB : \#CSB : \#MSB$, in which $\#LSB$, $\#CSB$, and $\#MSB$ are the numbers of free LSB, CSB, and MSB pages within the SSD during the runtime, respectively. The sUB scheme is always used as a complement to other non-utilization-based schemes, e.g., sHG, sSB, and sQD.

One of the seven schemes or their combinations can be deployed for specifying page-types. They also can be simply integrated into a practical SSD. In runtime, a specific scheme can be dynamically chosen according to current application scenes. For instance, multi-queue SSD [14], where I/O streams are distinguishable because multiple host I/O queues are exposed to the SSD directly, can perform different schemes for different streams. Besides, the write operations generated by garbage collection usually are specified by sUB scheme for purpose of balancing the utilization of different types of pages.

4.3 Type-Specified Page Allocation

In PA-SSD, the user write requests are assigned appropriate page-types by the host interface according to the page-type specifying schemes described above. The assigned page-types to user write requests are inherited by their page-sized sub-requests in the subsequent page allocation process. This is in contrast to the type-blind page allocation strategy in the conventional TLC SSD. Moreover, the design of a lone candidate page per plane in conventional TLC SSD (with more details in Section 2) greatly limits the flexibility of allocating a desired type of page for a sub-request. Accordingly, in this subsection, we first discuss how PA-SSD provides multiple candidate pages with different page-types within each plane, then we present the redesigned *BLAlloc* primitive in PA-SSD, namely *pa-BLAlloc*, to realize type-specified page allocation.

4.3.1 Providing Multiple Candidate Pages within a Plane

In the conventional SSD design, there is only one active block in each plane, and there is only one candidate page in the active block because of the strict program order within a block, which leads to the design of a lone candidate page per plane. While this design significantly simplifies the management of flash resources, it severely limits the ability to allocate type-specified pages. In fact, after the *PLAlloc* primitive determines the channel ID, chip ID, die ID, and plane ID used for serving a write sub-request, there is no other choice but the one single candidate page in the candidate plane that can be allocated for serving the sub-request. To address this problem, one possible solution is to modify the *PLAlloc* primitive to first select a candidate plane that has the assigned type of page as candidate page to meet the page-type requirement. This solution, however, means a fix path from channel all the way to plane, which severely limits the selection of channel, chip, die, and plane for the purpose of exploiting hardware

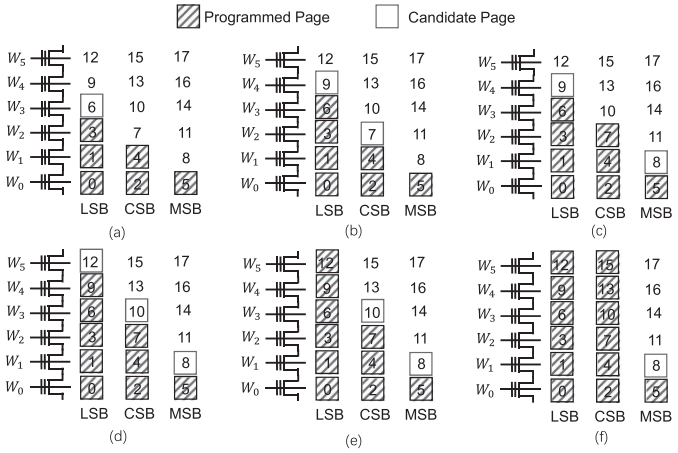


Fig. 8. Candidate pages within an active block in PA-SSD. In (a), only the next LSB page is a candidate page; in (b), both the next LSB page and CSB page are candidate pages; in (c), both the next LSB page and MSB page are candidate pages; in (d), all of the next LSB page, CSB page, and MSB page are candidate pages; in (e), the LSB pages are used up, leaving only CSB and MSB pages for serving subsequent sub-requests; in (f), both the LSB and CSB pages are used up, leaving only the slowest MSB pages for serving subsequent sub-requests.

parallelism [8], [10], significantly confining performance potentials. Thus, we prefer providing more candidate pages within each and every plane to searching a suitable plane to match the page-type requirement of sub-requests in designing PA-SSD, by *appropriately relaxing program constraints within blocks* and *providing multiple active blocks within each plane*.

① *Relaxing Program Constraints within Blocks.* The strict program order within blocks in the conventional TLC SSD design minimizes the inter-page (inter-cell) program interference by guaranteeing that a fully-programmed wordline is interfered by only one adjacent page programming [16]. For instance, in Fig. 5, before programming page 11 in $Wordline_2$, all pages adjacent except page 14 have been programmed, thus, the fully-programmed $Wordline_2$ only suffers from adjacent page programming by page 14. Another benefit of the strict program order is that LSB page is guaranteed to be programmed the first while MSB page is guaranteed to be programmed the last within a wordline. However, the strict program order actually is not essential for providing these two guarantees or benefits [18], [20].

For purpose of providing multiple candidate pages within a plane, PA-SSD manages pages within block with the following three program constrains instead of the strict program order used in conventional SSD:

- LSB, CSB, and MSB pages are programmed in order of their respective IDs, respectively;
- A CSB page can be programmed only when LSB pages in adjacent wordlines have been programmed;
- An MSB page can be programmed only when CSB pages in adjacent wordlines have been programmed.

By using these program constraints to manage pages, PA-SSD provides more than one candidate page with different types within an active block in the vast majority of cases, as illustrated by the examples shown in Fig. 8.

However, there are two main concerns with employing this page management in TLC blocks, namely, the cell-to-cell program interference and the memory space required for buffering un-fully-programmed wordlines.

For the first concern, the three program constrains actually provide guarantee that fully-programmed wordlines are interfered by no more than one adjacent page programming, which is the core of strict program order to minimize cell-to-cell program interference, thus, they theoretically provide the same benefit as strict program order. In addition, similar studies presented in [20] and [18] have experimentally demonstrated that using relaxed program constraints to approximate the two guarantees within MLC blocks does not significantly increase program interference errors. Because of the very similar characteristics of cell-to-cell program interference in TLC flash to that in MLC flash [16], we propose to use such similar relaxed program constraints in TLC flash with the justifiable stipulation that program interference errors are negligible.

For the second concern, as discussed in Section 2, conventional TLC SSD usually buffers the LSB and CSB pages within un-fully-programmed wordlines to shorten the latency of programming of CSB and MSB pages. In the PA-SSD design, as pages within a block are programmed with relaxed program constraints, multiple wordlines are un-fully-programmed during the runtime, which makes it costlier, if not impractical, to use a very large buffer for these un-fully-programmed wordlines. Therefore, in PA-SSD, the un-fully-programmed wordlines are not buffered, but at the possible expense of increased program latencies for CSB and MSB pages. Fortunately, our tests show that the negative impact is negligible primarily because of the significantly reduced *slow writes* in PA-SSD.

②*Providing Multiple Active Blocks within Planes.* Generally, in the conventional TLC SSD design, each plane maintains only one active block for serving subsequent write sub-requests. Only when the free pages within the active block are exhausted, the block is deactivated and another block with free pages is activated as the new active block. This design of one active block per plane is simple and effective but not optimal when the strict program order within blocks is replaced by our relaxed program constraints. For instance, when the active block runs out of the fast pages, it can serve subsequent sub-requests only with the slow pages, as illustrated by the examples depicted in both Figs. 8e and 8f.

To further increase the flexibility of allocating a desired type of page within a candidate plane, the PA-SSD manages blocks within plane with a fine-grained manner that blocks are partitioned into four pools and three active blocks, as depicted in Fig. 9. Specifically, the four pools are *LSB candidates' pool*, *CSB candidates' pool*, *MSB candidates' pool*, and *used pool*, while the three active blocks are *active LSB block*, *active CSB block*, and *active MSB block*. The *LSB candidates' pool* contains all erased blocks, blocks with all and only LSB pages programmed are belonged to the *CSB candidates' pool*, blocks with all and only MSB pages free are belonged to the *MSB candidates' pool*, and the *used pool* contains blocks with all pages programmed. The three active blocks are obtained from their corresponding candidate pools and are used for serving writes with corresponding page-type. The *active LSB block* is deactivated and inserted into *CSB candidates' pool* when its LSB pages are used up. Likewise, the *active CSB block* and the *active MSB block* are finally inserted to

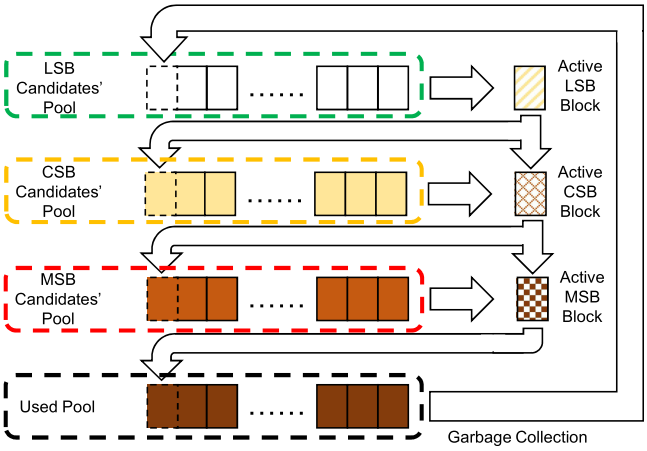


Fig. 9. Block management in PA-SSD. Blocks are partitioned into four pools, namely *LSB candidates' pool*, *CSB candidates' pool*, *MSB candidates' pool*, and *used pool*. Three active blocks, i.e., *active LSB/CSB/MSB block*, are provided simultaneously for serving write sub-requests.

MSB candidates' pool and *used pool*, respectively. GC operation reclaims fully programmed blocks from the *used pool* to the *LSB candidates' pool*. The *CSB candidates' pool* may be empty and thus it cannot offer new *active CSB block* in some situations, then the *active LSB block* also is used as *active CSB block* for serving write sub-requests allocated with both LSB page and CSB page. Likewise, the *active CSB block* also may be used as *active MSB block* simultaneously.

4.3.2 pa-BLAlloc *Primitive*

By utilizing the new page management and block management described in Section 4.3.1, PA-SSD is able to provide up to three candidate pages with different types within a candidate plane, largely increasing the flexibility to allocate pages for sub-requests with respect to their assigned page-types. However, the original *BLAlloc* primitive could not exploit this benefit when allocating pages within plane.

In conventional SSD design, the *BLAlloc* primitive of the page allocation strategy selects the lone active block within the candidate plane as the target block, and selects the lone candidate page within the block mandated by the strict program order as the target page. In PA-SSD design, in contrast, there are up to three active blocks within a plane and up to three candidate pages within an active block. This provision of PA-SSD makes it possible for its redesigned primitive to select candidate pages for sub-requests according to their assigned page-types. To this end, PA-SSD proposes the *pa-BLAlloc* primitive that repurposes the conventional *BLAlloc*

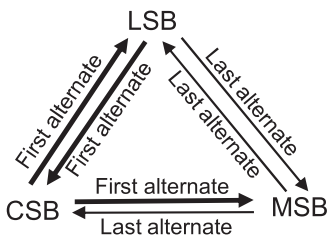


Fig. 10. Exception handling. When a specified type of page is not available in the candidate plane, *pa-BLAlloc* allocates the first alternate type of page if it is available, otherwise, it allocates the last alternate type of page. In the figure, A→Bmarked with First/Last alternate means A is the first/last alternate of B.

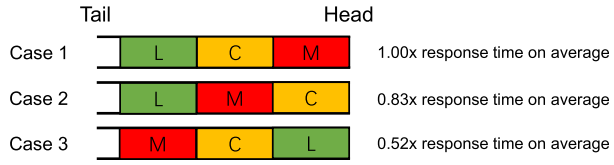


Fig. 11. The execution order of write sub-requests influencing the average write latency significantly. The average write response time of the three cases are normalized to that of Case 1.

primitive to allocate block IDs and page IDs for write sub-requests. Specifically, after the channel ID, chip ID, die ID, and plane ID are determined by the *PLAlloc* primitive, a candidate plane for serving the write sub-request is determined. Then, *pa-BLAlloc* parses the sub-request to obtain the assigned page-type, and selects a corresponding active block as the target block. Finally, it selects the specified type of candidate pages in the target block and returns their IDs.

Though the design of multiple candidate pages per plane in PA-SSD makes it more likely to allocate a specified-type of page for serving a write sub-request, there is no guarantee that the page-type requirement will be met. For instance, if the *active MSB block* is in the state depicted in Fig. 8b, and a sub-request with the specified page-type of MSB is supposed to be served by this plane, then the requirement cannot be met because no MSB candidate page is available. Although our experimental results reveal that over 98 percent write sub-requests are successfully allocated with their specified-type of pages in PA-SSD (detailed in Section 6.1), *pa-BLAlloc* also must be able to handle the rare exceptions where a required type of page is not available.

When the required page-type of a sub-request is not available, *pa-BLAlloc* allocates another type of candidate page within the plane for serving the sub-request based on the exception handling policy described in Fig. 10. That is, *pa-BLAlloc* allocates the first alternate type of page if it is available, otherwise, it allocates the last alternate type of page. Especially, MSB page is always the last alternate for the other two page-types due to its long program latency.

4.4 Page-Type Aware Sub-Request Scheduling

Once a type-specified page has been allocated, the write sub-request is inserted into its corresponding chip's queue, to wait for actual execution by the flash chip controller. The sub-requests in queue, usually including write, read, and erase operations, can be further reordered by TSU for specific purposes, as described in Section 2, such as FCFS and RP, and other advanced policies. Nevertheless, these policies actually do not take page-types into account. In the following, we first explain the benefit of scheduling write sub-requests according to page-types, and then detail the corresponding Page-type Aware Sub-request scheduling policy in the TSU of PA-SSD.

4.4.1 *Why Scheduling Write Sub-Requests According to Page-Type?*

Fig. 11 demonstrates the impact of execution order on write response time under three typical cases, each of which is a different execution order of three write sub-requests (or program operations equivalently). In the first case, an MSB programming is followed by a CSB programming and then

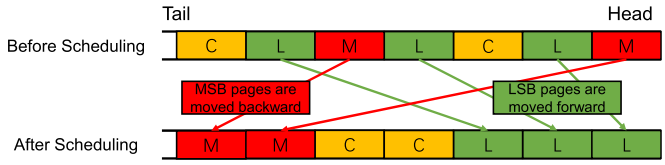


Fig. 12. TSU scheduling policy in PA-SSD. The write sub-requests allocated with LSB pages are moved to the head of the queue while those allocated with MSB pages are moved to the tail of the queue.

an LSB programming. Assuming that an LSB programming, a CSB programming, and an MSB programming consume 500 μs, 2000 μs, and 5500 μs, respectively, then, the average response time of three sub-requests in this case is 7000 μs. For the second case, the CSB programming is executed first, followed by an MSB programming and then an LSB programming. For this case, the average response time is 5833 μs, which is 16.7 percent shorter than that of the first case. And for the third case, the LSB programming is executed first, followed by a CSB programming and then an MSB programming, resulting in 3667 μs response time on average, which is 47.6 percent shorter than that of the first case and is 37.1 percent shorter than that of the second case. In short, these three typical cases reveal the fact that an execution order of write sub-requests has great impact on the write performance.

We further introduce a mathematical model to quantitatively analyze the impact of write sub-requests' execution order on average write response time.

Considering a queue composed of n write sub-requests, each of which has l_i (for i in $[1, n]$) program latency. As the execution of sub-requests in the back of the queue should wait for those in the front of the queue, the response time of the kth sub-request is therefore calculated as $\sum_{i=1}^{k} l_i$. Accordingly, the average response time of the n sub-requests in the queue, denoted as Avg_Res_Time, is calculated as:

$$Avg\_Res\_Time = \frac{1}{n}\sum_{k=1}^{n}\left(\sum_{i=1}^{k} l_i\right) = \frac{1}{n}\sum_{i=1}^{n}[(n-i+1)*l_i]. \tag{1}$$

From the Equation (1), the program latency of a sub-request with smaller subscript i (which means that the sub-request is closer to the queue head) has a higher weight, i.e., $(n-i+1)/n$, on impacting the average response time. This is because a sub-request close to the queue head has much greater impact on delaying subsequent sub-requests than those close to the queue tail. According to this analysis, moving the sub-requests with low write latency (i.e., LSB pages) to the queue head while moving those with high write latency (i.e., MSB pages) to the queue tail helps to reduce the average response time. This actually inspires designing the scheduling policy in TSU of PA-SSD.

4.4.2 TSU Scheduling Policy in PA-SSD

As described in the previous subsection, scheduling write sub-requests according to their page-types can reduce the average write response time. However, a conventional SSD has a strict programing order, as described in Section 2, which actually limits scheduling space. Take the program order depicted in Fig. 5 as an example, the program of page 5 (MSB page) must be programmed before page 6 (LSB page), thus, a sub-request allocated with page 6 cannot be executed before a sub-request allocated with page 5. Besides, an SSD without our proposed page-specifying mechanism cannot benefit from this scheduling because a write request comprising different page-type sub-requests actually cannot be accelerated as a whole.

PA-SSD relaxes the program constraints within blocks, as described in Section 4.3.1. Therefore, LSB programming always can be scheduled to execute ahead of CSB and MSB programming. Likewise, CSB programming always can be scheduled to execute before MSB programming. Besides, PA-SSD allocates the same type of pages for sub-requests of a write request, can eliminate the performance difference of these sub-requests. These two respects make a page-type aware sub-request scheduling effective.

The scheduling policy introduced in PA-SSD is named PAS, as be illustrated in Fig. 12. Specifically, a sub-request allocated with LSB page has the highest priority in three pages; a sub-request allocated with CSB page has higher priority than those allocated with MSB pages. To avoid sub-requests with lower priority being starved, PAS defines two thresholds for MSB and CSB pages, respectively. The thresholds are the numbers of preemptive pages. When these two thresholds are too low, few sub-requests can be scheduled by PAS to reduce the average write response time. On the other hand, when these two thresholds are too high, the average write response time can be reduced but the write response time of sub-requests allocated with CSB/MSB pages can be increased.

PAS is only responsible for scheduling write sub-requests. That is, it does not schedule read sub-requests and erase operations within a chip's queue. In some SSDs, write, read, and erase operations have their own independent queues. In these situations, PAS is merely applied to write sub-request queues without affecting other queues. In other SSDs that write, read, and erase operations share the same queues, the PAS policy can only exchange write sub-requests and work with other policies, such as FCFS and RP for scheduling read sub-requests and erase operations. For instance, when integrated with RP, read sub-requests are scheduled to the front end while erase operations are scheduled to the back end of the queue, then the PAS policy handles the write sub-requests in the queue.

4.5 Implementation Costs

Compared with baseline SSD design that allocates pages without considering page-types, PA-SSD introduces extra computation and memory consumption for allocating pages judiciously according to their page-types and tracking free pages within blocks. As PA-SSD differs from conventional SSD in the design of host interface, FTL, and TSU, in the following, we detail the costs of PA-SSD from the perspectives of these three components, respectively.

For the host interface, PA-SSD specifies a page-type for each sub-request. To this end, PA-SSD first determines a page-type according to a deployed page-type specifying scheme, e.g., sQD, and then inserts a tag to a sub-request for indicating the determined page-type. As elaborated in Section 4.2, our proposed page-type specifying schemes determine page-types are with straightforward logic, and their computational complexities are $O(n)$ (n represents the

TABLE 1
Configurations of the Simulated SSD Device

Parameters		Values	Parameters		Values
Flash Type		TLC	Structural	Chips per Channel	2
Latencies	Transfer	3ns/Byte		Planes per Chip	16
	Read	100μs		Blocks per Plane	384
	Program	500μs (L) 2000μs (C) 5500μs (M)		Pages per Block	384
				Page Size	8KB
				Over-Provision	15%
	Erase	15ms		GC Threshold	30%

number of user write requests). On the other hand, the tag for indicating a page-type needs only two bits, which is negligible compared with 8 KBytes DRAM for caching a user write sub-request.

For the FTL, PA-SSD utilizes fine-grained block management and page management that result in extra memory usage. Specifically, PA-SSD records the most recently programmed LSB page, CSB page, and MSB page IDs (each ID with 2B) for each block. Considering a 288 GB TLC SSD with structure parameters listed in Table 1, it needs extra 576 KB (=8Channels * 2Chips/Channel * 16Planes/Chip * 384Blocks/Plane * 6 Bytes/Block).

For the TSU, PA-SSD utilizes PAS for scheduling write sub-requests, it reorders write sub-requests according to their page-types. Actually, PAS has the same computational complexity as RP (read-priority) that reorders sub-requests according to Read/Write type.

In summary, the computational and memory cost of implementing PA-SSD are acceptable comparing with its benefits on improving write/read performance, detailed in Section 6.

5 Experimental Setups

5.1 Simulator

We implement PA-SSD based on SSDSim, which is a popular simulator whose accuracy has been validated against a real hardware platform [10]. To accommodate our requirement for evaluating the performance of PA-SSD, we have made the following major modifications to SSDSim:

- We modified the write latency calculation method to support the diverse program latencies in TLC flash.
- We introduced page-type specifying schemes to assign page-types for user write requests.
- We modified the page allocation strategy to support type-specified page allocation.
- We realized the page-type aware sub-request scheduling policy to reorder sub-requests in chips' queues.
- We introduced a multi-run mode for replaying the same trace multiple times to observe long-term performance.

In our experiments, we simulated a 288 GB TLC SSD with 8 channels. In Table 1, we list the configuration details of the simulated SSD. In PA-SSD, the program latencies of CSB and MSB pages are lengthened by the amount of time taken to read one and two pages respectively, as the cost of not using un-fully-programmed wordline buffer. Besides, to reflect the impact of garbage collections, before replaying traces, the simulated SSD is aged to that with 70 percent of its capacity being used (corresponds to the GC threshold as 30 percent).

TABLE 2
Characteristics of the Tested Workloads

Name	Request Count	Write Req. Ratio	Read (GB)	Write (GB)	Avg. W. Size (page)	Large Write Ratio
tFin	5,334,987	77%	2.7	14.6	0.5	6%
tRA	2,214,073	90%	2.3	15.6	1.0	12%
tDAP	1,086,729	44%	36.2	44.1	12.1	32%
tRBESS	5,250,996	82%	97.0	47.9	1.5	30%
tDTR	18,195,701	32%	252.1	176.1	3.9	53%
tExch	7,450,837	46%	37.4	41.3	1.6	17%
tMSFS	29,345,085	33%	200.9	102.3	1.4	10%
tBS	11,873,555	49%	149.8	166.2	3.7	67%

5.2 Workloads

To fully evaluate the performance of PA-SSD, we use 8 real application workloads in our experiments. The statistics of these workloads are listed in Table 2. Among them, the tFin is a light weight workload collected from on-line transaction processing applications running at financial institutions [21], while the other workloads spanning from lightweight to heavyweight are collected from Microsoft Production Server [22].

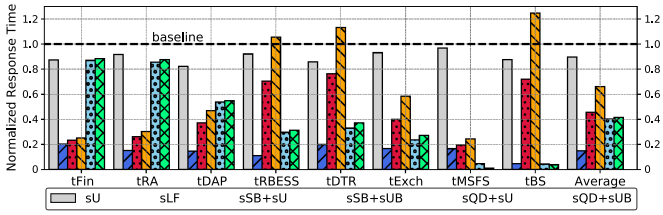
When a write request involves more than one page-sized sub-request, we regard the request as a large write. Accordingly, we record the ratio of large write of the workloads as the number of large write requests divided by the total number of write requests, as the last column of Table 2. The average write size of the workloads are reported as the number of pages in the sixth column of Table 2.

5.3 Evaluated Page-Type Specifying Schemes

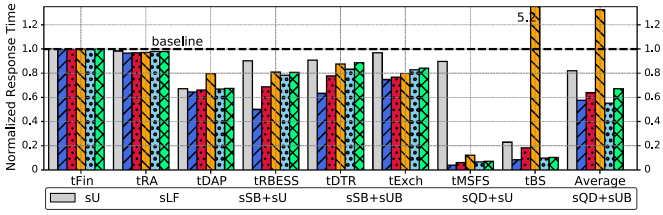
In our experiments, we evaluated six page-type specifying schemes, namely, *sU*, *sLF*, *sSB+sU*, *sSB+sUB*, *sQD+sU*, and *sQD+sUB*. The details of these schemes are presented in Section 4.2. Specifically, the notations sSB+sU, sSB+sUB, sQD+sU, and sQD+sUB indicate paired-schemes, in which the second scheme is used as a complement to the first scheme. Take sQD+sUB as an example, PA-SSD assigns LSB pages for write requests when the device-level I/O queue is longer than a preset threshold; Otherwise, it assigns page-types for write requests according to the sUB scheme. The queue depth threshold of the sQD scheme is set to 10 requests in our experiments. For the sSB scheme, requests with only one sub-request involved are assigned the LSB pages while large writes are assigned by a complementary scheme. The sHG scheme is not evaluated because the tested traces do not offer response time requirement of the requests. In addition, as no write buffer is utilized in our simulated SSD, the sWB scheme is in fact reduced to the sLF scheme.

To validate the effectiveness of PAS scheduling policy in PA-SSD, we also evaluated the performance of five specifying schemes augmented with PAS, which are denoted as *sU+PAS*, *sSB+sU+PAS*, *sSB+sUB+PAS*, *sQD+sU+PAS*, and *sQD+sUB+PAS*, respectively. The two thresholds in PAS for avoiding starvation of sub-requests with CSB and MSB pages are set as 10 and 20, respectively.

The conventional SSD design that allocates pages and schedules sub-requests without considering the page-types is used as the baseline system for evaluating the performance



(a) Average write response time, normalized to that of the baseline.



(b) Average read response time, normalized to that of the baseline.

Fig. 13. Performance evaluation of PA-SSD. Workloads on the x-axis are ordered in their relative intensity from light (left) to heavy (right).

of PA-SSD with various page-type specifying schemes and PAS scheduling policy.

6 Experimental Results

In this section, we report and discuss the experimental evaluation results of PA-SSD.

6.1 Average Response Time without PAS

In this subsection, we use the average response time, normalized to that of the baseline, as a measure to evaluate the performance of PA-SSD with various page-type specifying schemes. The benefit of PAS is detailed in the next subsection (Section 6.2), thus we just highlight the benefits of allocating pages according to various specifying schemes in this subsection. The results on the average write and read response times are shown in Figs. 13a and 13b, respectively. The workloads on the x-axis are sorted in their relative intensity increasing from light (left) to heavy (right).

The PA-SSD with sU scheme reduces the write response time and read response time by 10 and 18 percent respectively on average. This performance improvement is attributed to a balanced proportion of *slow writes* and *fast writes* as PA-SSD with sU effectively increases the *fast writes* to 34 percent while decreases the *slow writes* to 33 percent, as shown in Fig. 14.

By prioritizing the LSB pages in allocation, sLF achieves significantly lower write and read response time than the baseline, by 85 and 42 percent respectively on average. The sLF scheme has the lowest write response time under most workloads except for the two heaviest workloads, i.e., tMSFS and tBS, where the LSB pages are rapidly depleted. The reasons behind this are elaborated in Section 6.3.

The sSB+sU scheme preferentially allocates LSB pages for small writes. Thus, for workloads with high proportions of small writes (low large write ratio), e.g., tFin and tMSFS, sSB +sU can greatly reduce the write response time. On average, sSB+sU reduces write response time and read response time of the baseline by 54 and 36 percent, respectively. On the other hand, the sSB+sUB scheme tends to allocate CSB and MSB pages for large writes because the LSB pages have been

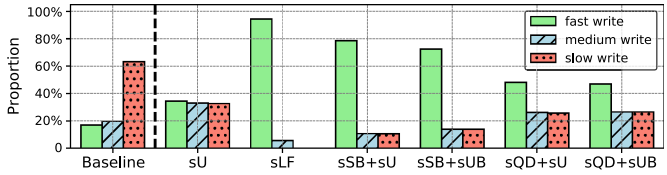


Fig. 14. The distribution of *fast/medium/slow writes* under the tDAP workload.

depleted for serving small writes, resulting in longer average response time than the baseline in some cases.

The sQD+sU scheme dynamically monitors the I/O intensity according to the length of I/O queue, and temporarily reduces waiting time for all requests by assigning LSB pages to all write requests during I/O bursts. For light workloads, e.g., tFin and tRA, the sQD+sU scheme has an average write response time similar to the sU scheme because the I/O queue in these workloads seldom exceeds the threshold. But for heavy workloads, especially for tMSFS and tBS, the sQD +sU scheme achieves extremely low write and read response times. In short, sQD+sU reduces write response time and read response time by 60 and 45 percent on average across these eight tested workloads, leading to 2.5 times and 1.8 times write and read performance over the baseline design, respectively. The sQD+sUB scheme achieves still lower write response time on the two heaviest workloads. Especially for tMSFS, sQD+sUB reduces write response time of the baseline by 99 percent. Noticeably, the write response time of sQD +sUB is 6 percent higher than that of sQD+sU under light workloads (i.e., from tFin to tExch), while its write response time is 56 percent lower than that of sQD+sU under heavy workloads (i.e., tMSFS and tBS). This performance difference between sQD+sU and sQD+sUB is explained in Section 6.3.

Besides, our results also demonstrate that all tested page-type specifying schemes and the baseline design experience nearly same numbers of garbage collections with less than 1 percent differences.

The performance improvement of PA-SSD without PAS over the baseline stems from its ability to effectively adjust the proportions of *fast writes*, *medium writes*, and *slow writes*. Fig. 14 illustrates the distribution of requests belonging to these three groups of write requests under the tDAP workload for the baseline design and the main PA-SSD schemes. In the baseline design, the proportion of *slow writes* is much higher than those of the other two groups (explained in Section 3). On the contrary, because of the type-specified page allocation for sub-requests, PA-SSD consistently keeps the proportion of *slow writes* the lowest among the three groups (not higher than 1/3). By employing non-utilization-based page-type specifying schemes, e.g., sLF, sSB, and sQD, PA-SSD further improves write performance by promoting the proportion of *fast writes* and reducing the proportion of *slow writes*. Besides, using sUB instead of sU as a complement to sSB and sQD always leads to more uniform distributions of the three groups.

We also demonstrate the efficiency of the type-specified page allocation strategy in PA-SSD by evaluating the success rate of PA-SSD on allocating specified-type of pages for sub-requests with various page-type specifying schemes. As shown in Fig. 15, the success rate is higher than 98 percent for five of the six evaluated schemes. The low success

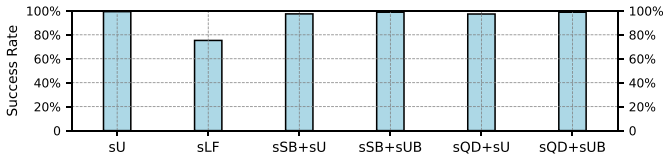


Fig. 15. The success rate of PA-SSD on allocating specified-type of pages for sub-requests.

rate of PA-SSD with sLF is due to its greedy on using LSB pages, detailed in the Section 6.3.

6.2 Benefit of PAS

We compare the average write response time of sU+PAS, sSB+sU+PAS, sSB+sUB+PAS, sQD+sU+PAS, and sQD+sUB +PAS to their respective counterparts as sU, sSB+sU, sSB +sUB, sQD+sU, and sQD+sUB, and the results are listed in Table 3.

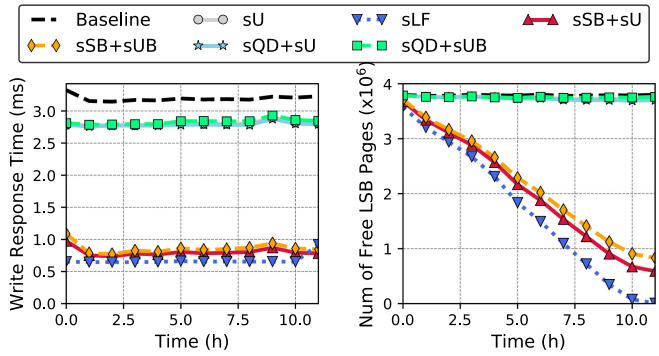
By integrating PAS scheduling policy, the write response time of PA-SSD with sU scheme can be reduced by 30 percent on average. Note that for lightweight workloads as tFin and tRA, the chips' queues usually contain only one or two sub-requests. Consequently, there are little benefits in these cases. On the contrary, for heavyweight workloads as tDTR, tExch, and tMSFS, PAS reduces over 40 percent write response time for PA-SSD with sU scheme.

The PAS scheduling policy works even better on PA-SSD with sSB+sU and sSB+sUB schemes as it reduces 40 and 41 percent write response time on average, respectively. Since under sSB scheme, small write requests with only one sub-request are assigned with LSB pages, reducing the response time of the sub-requests will absolutely reduce the response time of the write requests.

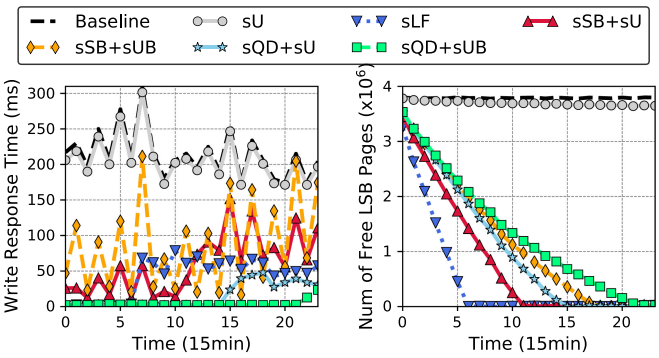
The benefits of PAS on reducing write response time of PA-SSD with sQD+sU and sQD+sUB schemes are relatively low, which are 11 and 17 percent reduction on average, respectively. Note that the length of chips' queue is positively correlated to the length of the device-level I/O queue. As a result, when the chips' queue is long under sQD scheme, most of the sub-requests actually are allocated with LSB pages. When most of the sub-requests within a chip's queue have the same type of pages, PAS seldom schedules the sub-requests. On the other side, when the chips' queue is short, PAS also does not work. Consequently, the reduction in write response time of PAS with sQD scheme is lower than other schemes. However, this does not mean

TABLE 3
Write Response Time Reduction by PAS

Workload	sU +PAS	sSB +sU +PAS	sSB +sUB +PAS	sQD +sU +PAS	sQD +sUB +PAS
tFin	0%	0%	0%	0%	0%
tRA	2%	7%	7%	2%	2%
tDAP	21%	25%	25%	13%	13%
tRBESS	60%	76%	71%	7%	7%
tDTR	40%	47%	33%	9%	30%
tExch	44%	67%	70%	22%	25%
tMSFS	42%	51%	78%	1%	11%
tBS	33%	50%	47%	38%	45%
Average	*30%*	*40%*	*41%*	*11%*	*17%*



(a) tFin



(b) tMSFS

Fig. 16. Write performance over time.

that PAS has little benefit under these two schemes. Actually, our results manifest that PAS helps greatly on smoothing the performance of PA-SSD. Specifically, the standard deviation of write response time in PA-SSD with sQD+sUB +PAS is 55 percent lower than that in PA-SSD with sQD +sUB, making PA-SSD with sQD+sUB+PAS not only fast but also stable.

6.3 Write Performance Over time

In this subsection, we analyze the write performance over time under two typical workloads, i.e., tFin (light) and tMSFS (heavy), to better understand the write performance characteristics of different page-type specifying schemes.

We first analyze the write response time under tFin, which is depicted in the left diagram of Fig. 16a. Because of the low I/O intensity and small ratio of large writes, even the baseline design can obtain a response time 21 percent higher than the average program latency in TLC flash (2.6 ms in our simulated SSD). By allocating pages for sub-requests with pages of the specified page-type, sU scheme reduces the write response time of the baseline by 13 percent. As the I/O queue under tFin seldom reaches the threshold of the sQD +sU and sQD+sUB schemes, their performances are very close to that of sU. On the contrary, the sLF scheme greatly reduces the write response time by fully benefiting from the LSB pages. However, it also makes the number of free LSB pages within the SSD decrease very fast, as shown in the right diagram of Fig. 16a. Especially, during the last hour of the simulation, there are no free LSB pages left. Although the garbage collection in SSD can continuously reclaim LSB pages in runtime, the pace with which the sLF scheme allocates the LSB pages is much faster than that of garbage collection to

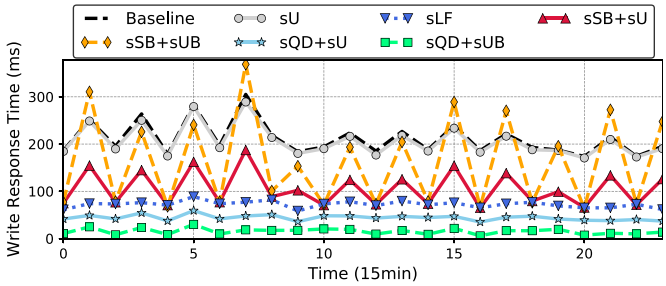


Fig. 17. Performance in Long-term under tMSFS workload.

reclaim the LSB pages, resulting in significant degradation in write performance at the end of the simulation.

For tMSFS, the sLF scheme achieves among the lowest response time during the first seven quarters (15 mins/quarter) of the simulation, which then gives way to quickly increased response time once free LSB pages are exhausted for the rest of the simulation, as shown in Fig. 16b. On the contrary, both the sQD+sU and sQD+sUB schemes maintain low write response times for a long period of time. Especially, sQD+sUB retains the low write response time for 22 quarters, which is three times that of the sLF scheme.

All of the PA-SSD schemes of sLF, sQD+sU, and sQD+sUB improve the write performance by effectively leveraging the LSB pages. Among them, sLF, as the most radical scheme, drastically reduces write response time once free LSB pages become available. However, once all free LSB pages are used up, the performance of subsequent I/O requests degrades noticeably. By dynamically monitoring and leveraging the I/O request queue depth, the sQD+sU and sQD+sUB schemes are able to use LSB pages judiciously and efficiently. As a result, although sQD+sU and sQD+sUB cannot achieve write response times as low as sLF on light workloads, they are capable of maintaining lower response times for longer periods of time under heavy workloads than the sLF scheme, achieving better overall performance. Additionally, by combining the sUB scheme that proactively reserves LSB pages during relative idle times, the sQD+sUB scheme performs even better than the sQD+sU scheme under the extremely heavy workloads, e.g., tMSFS and tBS.

6.4 Performance in Long-Term

As described above, the performance of sLF, sQD+sU, and sQD+sUB is highly sensitive to the number of free LSB pages within the SSD. When the free LSB pages are used up and even though garbage collection reclaims LSB pages in the runtime, the write performance of PA-SSD with these three schemes degrades. It is possible to introduce some methods to avoid or mitigate the condition in which free LSB pages are used up, for example, triggering background garbage collections during idle time to proactively reclaim LSB pages. A challenging question is, can these three schemes still provide a better write performance than the baseline if the free LSB pages are used up during busy time periods? To answer this question, we present the long-term performances of these schemes here to provide some insight.

We replay the tMSFS workload multiple times continuously and observe that the free LSB pages in PA-SSD with the sLF, sQD+sU, and sQD+sUB schemes are all used up before the third simulation. Therefore, we use the performance in the third simulation under tMSFS to demonstrate the long-term performance of PA-SSD, which is illustrated in Fig. 17. As the figure shows, even when the free LSB pages are used up, PA-SSD with sLF, sQD+sU, and sQD+sUB still outperforms the baseline significantly. Specifically, PA-SSD with the sLF, sQD+sU, and sQD+sUB schemes achieve 66, 79, and 93 percent lower write response times than the baseline, respectively. Particularly, this long-term performance analysis suggests a great potential of the PA-SSD with the sQD+sUB scheme in providing high and stable write performance in I/O-intensive environments.

7 Related Works

Because of the long program latencies of MLC and TLC flash, there have been many studies on improving the write performance of MLC/TLC SSD. We categorize related techniques for improving the SSD write performance into three major classes as follows.

① *Exploiting Parallelism.* The resources within the flash chip array are organized in a highly parallel architecture. It is beneficial to exploit this parallelism for improving write performance of SSD. Existing techniques generally exploit parallelism in two ways, improving the *PLAlloc* primitive to stripe sub-requests across channels, chips, dies, and planes [8], [10] and scheduling user write requests to increase flash resource utilization [9], [13], [19]. These techniques actually are compatible with and orthogonal to our proposed PA-SSD on improving write performance.

② *Using SLC Flash as Buffer.* As the write latency of SLC flash is much lower than that of MLC/TLC flash, many researchers and developers suggested employing SLC flash in MLC/TLC SSD as a write buffer to improve the write performance. In [23], L. Chang proposed a hybrid SSD, which combined with SLC chips and MLC chips, in which SLC chips serve hot data while MLC chips serve cold data. M. Murugan and D. Du proposed Hybrot, for avoiding the situation that SLC zone be worn out faster than MLC zone in a hybrid SSD by intelligently forward data flow to SLC zone and MLC zone [24]. In [4], S. Lee, et al. proposed FlexFS that enables SLC mode in MLC flash to improve write performance of MLC SSD. FlexFS introduces techniques as background migration that triggers data migration during idle time, and dynamic allocation that reduces the amount of data written to SLC zone, to reduce migration overhead in hybrid SSD. S. Im and D. Shin proposed ComboFTL that develops intelligent hot/cold data detection technique to improve performance and lifespan of a hybrid SSD [5]. W. Wang, et al., discussed the impact of the ratio of SLC zone and MLC zone within a hybrid SSD, and demonstrated that there is an optimal ratio for each workload to achieve the best performance [6]. In [7], C. Chang, et al., proposed SLO-aware Morphable SSD that adaptively changes SLC mode and MLC mode in MLC flash to meet service-level objectives. It decides a user write request should be processed in SLC mode or MLC mode in the runtime by analyzing SLO requirements of all requests in the queue. In [1], D. Sharma revealed the poor write performance of TLC flash and suggested an SLC-TLC combined hybrid SSD, where data are first written to SLC and then be migrated to TLC.

All these techniques develop hybrid SSD in one of two ways, i.e., introducing extra SLC flash chips [23], [24] or enabling the SLC mode in part of the MLC/TLC flash [1], [4], [5], [6], [7]. While the first approach increases the total SSD cost, the second reduces the total actual storage capacity. More importantly, the introduction of an SLC buffering layer in SSD can noticeably complicate the FTL implementation due to data migration between the SLC area and the MLC/TLC area [25]. In addition, the limited size of the SLC area can be filled up quickly during write intensive periods, leading to degraded performance for subsequent write requests. Differently, our proposed PA-SSD improves write performance without either sacrificing any storage capacity or significantly complicating the FTL implementation.

③*Taking Advantage of LSB Pages.* The low program latency of the LSB pages in MLC/TLC flash is exploited for improving write performance. In [3], Grupp et al. proposed a technique that proactively uses LSB pages to serve burst writes for improving peak performance. However, constrained by the strict program order within MLC/TLC flash blocks, the benefit of this technique is limited. In [18], Park et al. proved that the strict program order within an MLC flash block is an overprovision for reducing cell-to-cell program interference, and further proposed flexFTL that uses relaxed program constraints within MLC flash blocks to provide more flexible use of LSB and MSB pages. The flexFTL technique adaptively allocates LSB pages for write sub-requests according to the write buffer utilization to improve write performance. Our technique, PA-SSD, also relies on relaxed program constraints within TLC flash blocks to provide flexible use of LSB, CSB, and MSB pages. In addition, some of our page-type specifying schemes also try to take advantage of LSB pages for improving write performance. However, there are two major differences between flexFTL and PA-SSD. First, PA-SSD coordinately allocates pages of the same type for the sub-requests of a given user write request, while flexFTL does not consider the dependence and correlation among sub-requests, which results in inefficient mixed-type page allocation (detailed in Section 3); Second, PA-SSD provides a rich set of schemes to satisfy various performance requirements.

8 Conclusion

In this paper, we first demonstrated the write inefficiency of the type-blind page allocation design in conventional TLC SSD. To eliminate this write inefficiency, we presented PA-SSD, a page-type aware TLC SSD design that improves write/read performance by exploiting the diversity of flash pages. Specifically, PA-SSD assigns the page-types for user write requests according to seven proposed page-type specifying schemes, and allocates pages for the corresponding sub-requests according to their assigned page-types, finally schedules the sub-requests within chips' queue according to their page-types. The program constraints within planes and the *BLAlloc* primitive are redesigned to realize the type-specified page allocation in PA-SSD. We implemented PA-SSD with SSDSim and evaluated its performance with various page-type specifying schemes. Our simulation results show that PA-SSD with the *sQD+sUB+PAS* scheme improves write and read performance by 2.6 times and 1.5 times over the conventional SSD design.

Acknowledgments

The authors would like to thank the anonymous referees for their valuable comments and helpful suggestions. This work is supported in part by NSFC No. 61872156, Creative Research Group Project of NSFC No. 61821003, National key research and development program of China (Grant No. 2018YFA0701804), the Fundamental Research Funds for the Central Universities No. 2018KFYXKJC037, Alibaba Group through Alibaba Innovative Research (AIR) Program, and the US NSF under Grant No. CCF-1704504 and CCF-1629625.

References

[1] D. Sharma, "System design for mainstream TLC SSD," in *Proc. Flash Memory Summit*, 2014, pp. 1–20.
[2] C. Matsui, T. Yamada, Y. Sugiyama, Y. Yamaga, and K. Takeuchi, "Tri-hybrid SSD with storage class memory (SCM) and MLC/TLC NAND Flash Memories," in *Proc. Flash Memory Summit*, 2017, pp. 1–22.
[3] L. M. Grupp, J. D. Davis, and S. Swanson, "The harey tortoise: Managing heterogeneous write performance in SSDs," in *Proc. USENIX Annu. Tech. Conf.*, 2013, pp. 79–90.
[4] S. Lee, K. Ha, K. Zhang, J. Kim, and J. Kim, "FlexFS: A flexible flash file system for MLC NAND flash memory," in *Proc. USENIX Annu. Tech. Conf.*, 2009, pp. 9–9.
[5] S. Im and D. Shin, *ComboFTL: Improving Performance and Lifespan of MLC Flash Memory using SLC Flash Buffer*. Amsterdam, Netherlands: Elsevier, 2010.
[6] W. Wang, W. Pan, T. Xie, and D. Zhou, "How many MLCs should impersonate SLCs to optimize SSD performance?" in *Proc. 2nd Int. Symp. Memory Syst.*, 2016, pp. 238–247.
[7] C.-W. Chang, G.-Y. Chen, Y.-J. Chen, C.-W. Yeh, P. Y. Eng, A. Cheung, and C.-L. Yang, "Exploiting write heterogeneity of morphable MLC/SLC SSDs in datacenters with service-level objectives," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1457–1463, Aug. 2017.
[8] A. Tavakkol, P. Mehrvarzy, M. Arjomand, and H. Sarbazi-Azad, "Performance evaluation of dynamic page allocation strategies in SSDs," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 2, pp. 1–33, Jun. 2016.
[9] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit.*, Feb. 2014, pp. 524–535.
[10] Y. Hu, H. Jiang, D. Feng, L. Tian, H. Luo, and S. Zhang, "Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity," in *Proc. Int. Conf. Supercomputing*, 2011, pp. 96–107.
[11] M. Jung, E. H. Wilson, and M. Kandemir, "Physically addressed queueing (PAQ): Improving parallelism in solid state disks," in *Proc. 39th Annu. Int. Symp. Comput. Archit.*, Jun. 2012, pp. 404–415.
[12] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. T. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 289–300, Jun. 2014.
[13] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve SSD performance," in *Proc. 22nd Int. Conf. Architectural Support Program. Languages Operating Syst.*, 2017, pp. 375–388.
[14] A. Tavakkol, M. Sadrosadati, S. Ghose, J. Kim, Y. Luo, Y. Wang, N. M. Ghiasi, L. Orosa, J. Gmez-Luna, and O. Mutlu, "FLIN: Enabling fairness and enhancing performance in modern NVMe solid state drives," in *Proc. ACM/IEEE 45th Annu. Int. Symp. Comput. Archit.*, Jun. 2018, pp. 397–410.
[15] Y. Li, C. Hsu, and K. Oowada, "Non-volatile memory and method with improved first pass programming," US Patent. US 8811091, Aug. 2014.
[16] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proc. IEEE*, vol. 105, no. 9, pp. 1666–1704, Sep. 2017.
[17] N. E. Workgroup, "Nvm express revision 1.3 specification," 2017. [Online]. Available: http://nvmexpress.org/
[18] J. Park, J. Jeong, S. Lee, Y. Song, and J. Kim, "Improving performance and lifetime of NAND storage systems using relaxed program sequence," in *Proc. 53nd ACM/EDAC/IEEE Des. Autom. Conf.*, Jun. 2016, pp. 1–6.

[19] B. Mao and S. Wu, "Exploiting request characteristics and internal parallelism to improve SSD performance," in *Proc. 33rd IEEE Int. Conf. Comput. Des.*, Oct. 2015, pp. 447–450.
[20] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation," in *Proc. IEEE 31st Int. Conf. Comput. Des.*, Oct. 2013, pp. 123–130.
[21] U. T. Repository, "Umass trace repository storage traces," 2018. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage/Storage
[22] S. I. Repository, "Snia block i/o traces," 2018. [Online]. Available: http://iotta.snia.org/tracetypes/3
[23] L.-P. Chang, "A hybrid approach to NAND-flash-based solid-state disks," *IEEE Trans. Comput.*, vol. 59, no. 10, pp. 1337–1349, Oct. 2010.
[24] M. Murugan and D. H. C. Du, "Hybrot: Towards improved performance in hybrid SLC-MLC devices," in *Proc. IEEE 20th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, Aug. 2012, pp. 481–484.
[25] F. Chen, T. Zhang, and X. Zhang, "Software support inside and outside solid-state devices for high performance and high efficiency," *Proc. IEEE*, vol. 105, no. 9, pp. 1650–1665, Sep. 2017.

Wenhui Zhang received the BS degree in mathematics from Wuhan University, in 2011. He is currently working toward the PhD degree of the Wuhan National Laboratory for Optoelectronics at the Huazhong University of Science and Technology. His research interests include erasure codes, storage systems, and parallel algorithms. He is a student member of the IEEE and ACM.

Qiang Cao received the BS degree in applied physics from Nanjing University, in 1997, the MS degree in computer technology, and the PhD degree in computer architecture from the Huazhong University of Science and Technology, in 2000 and 2003, respectively. He is currently a full professor of the Wuhan National Laboratory for Optoelectronics at the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a senior member of China Computer Federation (CCF) and the IEEE and a member of ACM.

Hong Jiang received the BSc degree in computer engineering from the Huazhong University of Science and Technology, Wuhan, China, in 1982, the MASc degree in computer engineering from the University of Toronto, Toronto, Canada, in 1987, and the PhD degree in computer science from the Texas A&M University, College Station, Texas, in 1991. He is currently chair and Wendell H. Nedderman Endowed professor of Computer Science and Engineering Department, University of Texas at Arlington. Prior to joining UTA, he served as a Program director at National Science Foundation (2013.1-2015.8) and he was at University of Nebraska-Lincoln since 1991, where he was Willa Cather professor of Computer Science and Engineering. He has graduated 16 PhD students who upon their graduations either landed academic tenure-track positions in PhD-granting US institutions or were employed by major US IT corporations. His present research interests include computer architecture, computer storage systems and parallel I/O, high performance computing, big data computing, cloud computing, performance evaluation. He recently served as an associate editor of the *IEEE Transactions on Parallel and Distributed Systems*. He has more than 300 publications in major journals and international Conferences in these areas, including the *IEEE Transactions on Parallel and Distributed Systems*, the *IEEE Transactions on Computers*, the *Proceedings of IEEE*, the *ACM Transactions on Architecture and Code Optimization*, the *ACM Transactions on Storage*, JPDC, ISCA, MICRO, USENIX ATC, FAST, EUROSYS, LISA, SIGMETRICS, ICDCS, IPDPS, MIDDLEWARE, OOPLAS, ECOOP, SC, ICS, HPDC, INFOCOM, ICPP, etc., and his research has been supported by NSF, DOD, the State of Texas and the State of Nebraska, and industry. He is a fellow of the IEEE, and Member of ACM.

Jie Yao received the BS degree in computer science and technology, and the MS and PhD degrees in computer architecture from the Huazhong University of Science and Technology, Wuhan, China, in 2001, 2004, and 2009, respectively. He is currently a lecturer with the Huazhong University of Science and Technology. His research interests include computer architecture, large scale storage systems, and performance evaluation. He is a member of China Computer Federation (CCF) and a member of the IEEE and ACM.