PA-SSD: A Page-Type Aware TLC SSD for Improved Write/Read Performance and Storage Efficiency

Wenhui Zhang[†]
Wuhan National Laboratory for Optoelectronics,
Huazhong University of Science and Technology
Wuhan, China
zhangwenhui@hust.edu.cn

Qiang Cao[*†]
Wuhan National Laboratory for Optoelectronics,
Huazhong University of Science and Technology
Wuhan, China
caoqiang@hust.edu.cn

Hong Jiang
Department of Computer Science and Engineering,
University of Texas at Arlington
Arlington, Texas
hong.jiang@uta.edu

Jie Yao[†]
Department of Computer Science and Technology,
Huazhong University of Science and Technology
Wuhan, China
jackyao@hust.edu.cn

ABSTRACT

TLC flash has three types of pages to accommodate the three bits in each TLC physical cell exhibiting very different program latencies, LSB (fast), CSB (medium), and MSB (slow). Conventional TLC SSD designs on page allocation to write requests do not take page types and their latency difference into consideration, missing on an important opportunity to exploit the potentials of fast writes.

This paper proposes PA-SSD, a page-type aware TLC SSD design, to effectively improve the overall performance by judiciously and coordinately utilizing the three types of pages on TLC flash when serving user write requests. 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. We achieve the PA-SSD design goal by addressing two key research problems: (1) how to properly determine page-type for each user write request and (2) how to allocate a physical page for each sub-request with an assigned page type from (1). For the first problem, seven page-type specifying schemes are proposed to investigate their effects under different workloads. On the other hand, we approach the second problem by redesigning the page allocation strategy in TLC SSD to uniformly and sequentially determine pages for allocation following the programming process of TLC flash. Under a wide range of workloads, our experiments show that PA-SSD can accelerate both the write and read performance without any sacrifice to storage capacity. Particularly, our proposed queue-depth based page-type specifying scheme improves write performance by 2.4 times and read performance by 1.5 times over the conventional TLC SSD.

CCS CONCEPTS

• **Hardware → External storage**;

KEYWORDS

TLC SSD, diverse program latencies, performance, page-type aware

ACM Reference Format:
Wenhui Zhang, Qiang Cao, Hong Jiang, and Jie Yao. 2018. PA-SSD: A Page-Type Aware TLC SSD for Improved Write/Read Performance and Storage Efficiency. In *ICS '18: 2018 International Conference on Supercomputing, June 12–15, 2018, Beijing, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3205289.3205319

1 INTRODUCTION

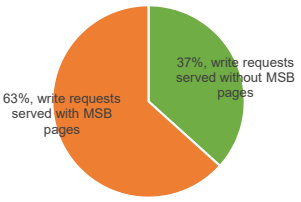
TLC (Triple-Level Cell) flash is gradually becoming a dominant storage media in Solid-State Drives (SSDs) because of its higher storage capacity and lower price per gigabyte than SLC (Single-Level Cell) flash and MLC (Multi-Level Cell) flash. However, TLC, which stores three data bits in each physical cell, requires finer-grained program steps, resulting in higher program latency than SLC and MLC flash [14, 19]. 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 design separates these bits into three types of pages with diverse program latencies, i.e., LSB pages, CSB pages, and MSB pages [7, 19]. Specifically, an LSB page has the shortest program latency (e.g., 500μs), a CSB page has the medium program latency (e.g., 2000μs), and an MSB page has the longest program latency (e.g., 5500μs) [7]. 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 [3, 9, 11, 19,

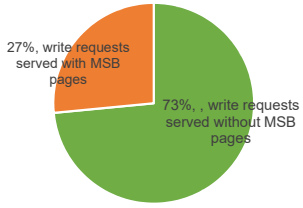
[*]Corresponding author.
[†]Key Laboratory of Information Storage System of Ministry of Education.

ICS '18, June 12–15, 2018, Beijing, China

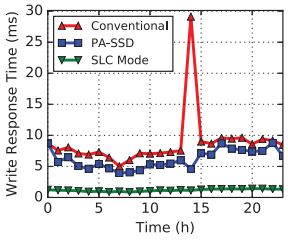
ACM ISBN 978-1-4503-5783-8/18/06…$15.00
https://doi.org/10.1145/3205289.3205319



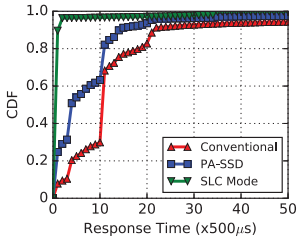
(a) A majority of user write requests are served with MSB pages in conventional SSD design.



(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.

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

21]. 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 [20], 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. 1(a), 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 coordinately allocate 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 and thus improve the 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, two challenging questions must be answered: Q1. How to determine which type of page to use for a given user write request? Q2. How to allocate pages to sub-requests of a user write request whose page-type has been specified (by an answer to Q1). For the first challenge, we propose seven page-type specifying schemes in PA-SSD, namely, the *US* (Uniformly Specification) scheme that uniformly (round-robin) assigns a page type to a request, the *HGS* (Host-Guided Specification) scheme that assigns a page type to a request according to response time requirement provided by the host, the *LFS* (LSB-First Specification) scheme that always assigns the LSB pages to any requests, the *SBS* (Size-based Specification) scheme that assigns a page type to a request according to the request size, the *QDS* (Queue-Depth-based Specification) scheme that assigns a page type to a request according to the device-level I/O queue depth, the *WBS* (Write-Buffer-based Specification) scheme that assigns a page type to a request according to the write buffer utilization, and the *UBS* (Utilization-based Specification) scheme that assigns page types according to their respective free capacities. 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. As shown in Fig. 1(b) (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. 1(a)), leading to lower write response time, as illustrated in Fig. 1(c) and Fig. 1(d).

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 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 *QDS* and *UBS* page-type specifying schemes, PA-SSD improves the write and read performances of the conventional TLC SSD by 2.4x and 1.5x on average, respectively.

The remainder of this paper is organized as follows. In Section 2, we present the necessary background of TLC SSD and related works on improving write performance 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 Section 5 and Section 6, we present our experimental setups and results for demonstrating the efficacy of PA-SSD. Finally, we conclude this paper in Section 7.

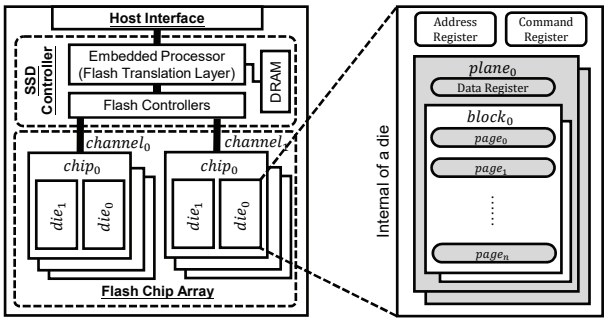


Figure 2: Architecture of SSD.

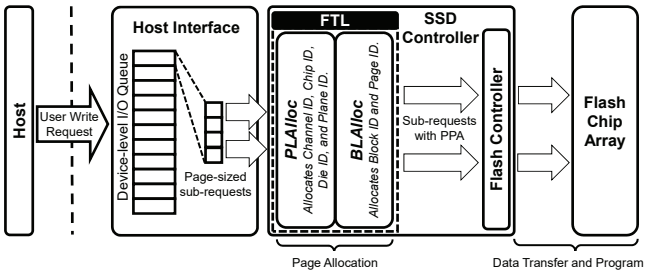


Figure 3: Write request execution workflow in a conventional SSD.

2 BACKGROUND AND RELATED WORKS

2.1 TLC SSD Preliminary

SSD architecture. As shown in Fig. 2, an SSD is composed of three components, i.e., host interface, SSD controller, and flash chip array [20]. The host interface supports communication between the host system and SSD controller, and maintains the device-level I/O queue [10]. 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, e.g., address translation and garbage collection. The SSD controller also communicates with the flash chip array through the flash controller. The flash chip array composed of multiple flash chips is connected to the flash controller via channels and provides the actual 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, 10, 20].

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, each with a specific LPA (*logical page address*) [3]. 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 [20]. 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 delivered to flash controllers where they are striped across channels/chips/dies/planes for parallel programming [20]. 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 [1], and a user write request is considered completed when all of its sub-requests are finished.

Page-types in TLC flash 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 [12, 19], 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 [7, 19]. Typically, for the 25nm TLC flash, LSB, CSB, and MSB pages exhibit 500μs, 2000μs, and 5500μs program latencies, respectively [7]. 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.

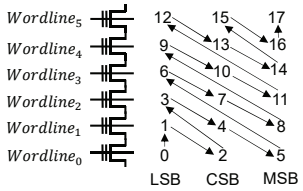


Figure 4: Program order of pages within a TLC block [1] [7].

Design of a lone candidate page per plane in conventional SSD. In the conventional SSD design, each plane maintains only one active block for serving subsequent page-sized write sub-requests. Furthermore, the pages within a TLC block should be programmed sequentially according to their IDs [1], as depicted in Fig. 4, for

[1] 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.

the purpose of reducing cell-to-cell program interference to fully-programmed wordlines. Therefore, 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* 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.

2.2 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, 20] and scheduling user write requests to increase flash resource utilization [6, 10, 13]. 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. This can be done in one of two approaches, i.e., introducing extra SLC flash chips [4, 15] or enabling the SLC mode in part of the MLC/TLC flash [3, 9, 11, 19, 21]. While the first approach increases the total SSD cost, the second reduces the total effective 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 [5]. 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. 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 [7], 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 [16], Park et al. proved that the strict program order within an MLC flash block is an over-provision 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 (detailed in Subsection 4.2).

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 [20]. A user write request completes when all of its constituent sub-requests are finished. Thus, the response time of a user write request 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., *LSB dominated writes*, *CSB dominated writes*, and *MSB dominated writes*, in order of increasing response time. Unfortunately, a large proportion of user write requests actually are *MSB dominated writes* in conventional TLC SSD, as illustrated in Fig. 1(a), 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 (*MSB dominated write*), denoted as P_{slow}, is equal to $(1-(\frac{2}{3})^n)$. On the other hand, the probability of all sub-requests being served by LSB pages (*LSB dominated 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 (*CSB dominated 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

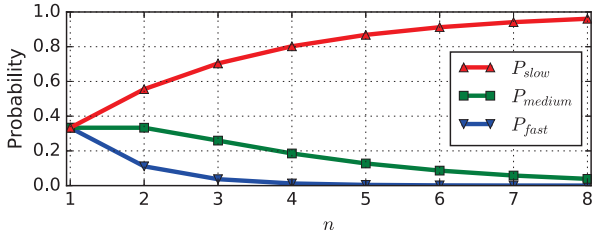


Figure 5: The probabilities of *LSB dominated write* (P_{fast}), *CSB dominated write* (P_{medium}), and *MSB dominated write* (P_{slow}) as a function of n (the number of sub-requests).

decrease with n, as illustrated in Fig. 5. For instance, when $n = 4$, $P_{slow} = 80\%$, meaning that most of the user write requests are *MSB dominated writes*. On the contrary, at $n = 4$, P_{fast} and P_{medium} decrease to 1% and 19%, respectively, diminishing the desirable impacts of *LSB/CSB dominated writes*.

This lopsided negative performance impact of MSB write dominance in conventional TLC SSD, where the undesirable *MSB dominated write* increases rapidly while the desirable *LSB dominated write* decreases rapidly with the request size n (number of sub-requests), motivates us to design a new page allocation strategy that minimizes *MSB dominated writes* while maximizes *LSB dominated 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 PA-SSD

4.1 Overview

The proposed PA-SSD is a **P**age-type **A**ware TLC SSD design, attempting to use the same type of pages for serving sub-requests of any single user write request. In Fig. 6, we illustrate the write request execution workflow in PA-SSD. There are two 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 Subsection 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 Subsection 4.3.

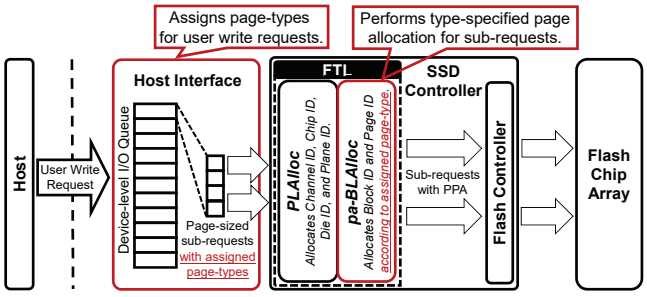


Figure 6: Write request execution workflow in PA-SSD. For each user write request, the host interface of PA-SSD proactively assigns a page-type that is inherited by all of its constituent page-sized sub-requests. The *BLAlloc* primitive is replaced by *pa-BLAlloc* that allocates pages for sub-requests according to their assigned page-types.

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 an *LSB(CSB/MSB) dominated write*. Thus, proactively determining the page-types for user write requests has the potential to adjust the ratios of *LSB/CSB/MSB dominated 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 (US). With this scheme, PA-SSD assigns the three page-types, i.e., LSB, CSB, and MSB, for user write requests 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 *LSB dominated writes*, *CSB dominated writes*, and *MSB dominated writes*.

Host-Guided Specification (HGS). 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' [22]. Accordingly, with the HGS 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 schemes, such as the US scheme. By employing HGS, 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 (LFS). With LFS, 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 [7, 16]. 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 Subsection 6.2). Thus, LFS is suggested as a turbo mode to improve write performance during write-intensive bursts.

Size-Based Specification (SBS). 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 [13]. Accordingly, with the SBS scheme, PA-SSD assigns LSB pages for small-sized requests (e.g., requests smaller than 8KB). On the other hand, for large-sized requests (e.g., requests larger than 8KB), PA-SSD determines their page-type by other schemes, e.g., US.

Queue-Depth-based Specification (QDS). 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 QDS 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., US and UBS schemes. The QDS scheme is regarded as a dynamic-LFS scheme that activates and deactivates LFS dynamically according to the I/O queue depth. In fact, QDS with queue depth threshold of 0 is reduced to the LFS scheme.

Write-Buffer-based Specification (WBS). In [16], 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 WBS scheme that assigns the LSB pages for user write requests when the write buffer is full. Compared with the QDS scheme that senses both write and read intensity from the device-level I/O queue depth, the WBS 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 WBS is employed. In fact, when the write buffer is very small, the WBS scheme actually becomes the LFS scheme.

Utilization-Based Specification (UBS). 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, UBS-based PA-SSD can effectively balance their utilizations. With UBS, 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 UBS 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 UBS scheme is always used as a complement to other non-utilization-based schemes, e.g., HGS, SBS, and QDS. Besides, the UBS scheme is also used as default scheme to specify page-types for write requests generated by garbage collection in PA-SSD.

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 Subsection 2.1) 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 [1], 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 parallelism [8, 20], 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 provides multiple active blocks within each plane.

The strict program order within blocks in the conventional TLC SSD design is necessary to minimize the inter-page (inter-cell) program interference by guaranteeing that a fully-programmed wordline is interfered by only one adjacent page programming. For instance, in Fig. 4, 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. But, the strict program order is not essential for providing these two guarantees or benefits [2] [16].

For purpose of providing multiple candidate pages within a plane while still providing the two guarantees, PA-SSD uses the following three program constraints within blocks to replace the strict program order:

- 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 instead of the strict program order, PA-SSD is able to provide 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. 7. However, there are two main concerns with employing these relaxed program constraints 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, similar studies presented in [2] and [16] 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

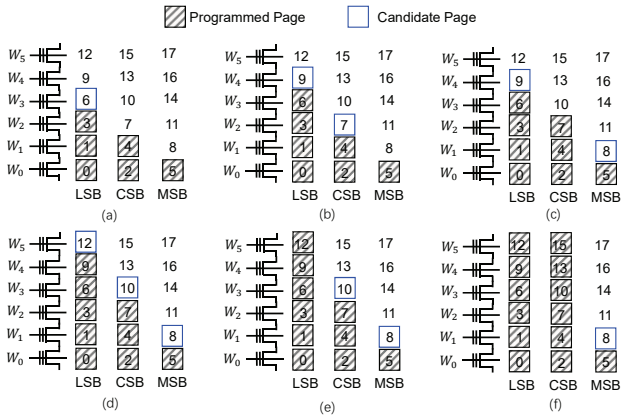


Figure 7: Candidate pages within an active block in PA-SSD. In (a), only the next LSB page is a candidate page; in (b), both the next LSB page and CSB page are candidate pages; in (c), both the next LSB page and MSB page are candidate pages; in (d), all of the next LSB page, CSB page, and MSB page are 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.

interference in TLC flash to that in MLC flash, we propose to use such similar relaxed program constraints in TLC flash with the justifiable stipulation that program interference errors will not be significant and acceptable. On the other hand, as discussed in Subsection 2.1, 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 *MSB dominated writes* in PA-SSD.

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 in both Fig. 7(e) and 7(f).

To further increase the flexibility of allocating a desired type of page within a candidate plane, the PA-SSD design maintains up to three active blocks per plane, which correspond to active blocks with free LSB pages, free CSB pages, and free MSB pages, respectively. In some cases, the three active blocks may actually be the same block, as our example depicted in Fig. 7(d), the block has free LSB pages, CSB pages, and MSB pages simultaneously. For the block depicted in Fig. 7(d), if page 12 as the last LSB page of the block is programmed, then it cannot provide LSB pages (just like the case depicted in Fig. 7(e)), therefore, another block in the plane should be activated for providing LSB pages while this block remains active for providing CSB pages and MSB pages. This multiple active blocks per plane design, which is similar to the multiple write points per chip design presented in [7], slightly complicates the FTL design and introduces storage overhead because more than one active blocks must be tracked by recording their runtime status. However, thanks to the powerful processor used in modern SSDs, this increased complexity is arguably negligible. On the other hand, as LSB, CSB, and MSB pages within a block in PA-SSD are programmed in order of their IDs, respectively, it is sufficient to track free pages by recording the IDs of the most recently programmed pages in each page-type, leading to at most 1152Bytes (=384block*3Bytes/block) storage overhead per plane.

By combining the design of multiple active blocks per plane and the design of multiple candidate pages per block, PA-SSD is able to provide up to three candidate pages with different types within a candidate plane, increasing the flexibility to allocate pages for sub-requests with respect to their assigned page-types.

4.3.2 ***pa-BLAlloc primitive.*** 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* 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 candidate active block of a plane for providing MSB pages is in the state depicted in Fig. 7(b), 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% write sub-requests are successfully allocated with their specified-type of pages in PA-SSD (detailed in Subsection 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

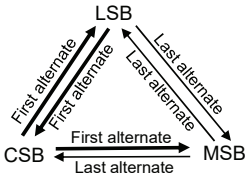


Figure 8: Exception handling. When a specified type of page is not available in the candidate plane, *pa-BLAlloc* allocates the first alternate type of page if it is available, otherwise, it allocates the last alternate type of page.

described in Fig. 8. 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.

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 [8]. 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 introduced a multi-run mode for replaying the same trace multiple times to observe long-term performance.

In our experiments, we simulated a 288GB 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% of its capacity being used (corresponds to the GC threshold as 30%).

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 (LSB) 2000μs (CSB) 5500μs (MSB)		Pages per Block	384
				Page Size	8KB
				Over-Provision	15%
	Erase	15ms		GC Threshold	30%

5.2 Workloads

To fully evaluate the performance of PA-SSD, we use 8 real application workloads in our experiments. These workloads are collected from [18] and [17], and their statistical characteristics are listed in Table 2. 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.

Table 2: Characteristics of the Tested Workloads

Name	Request Count	Write Request Ratio	Read (GB)	Write (GB)	Average Write Size (page)	Large Write Ratio
Fin	5,334,987	77%	2.7	14.6	0.5	6%
RA	2,214,073	90%	2.3	15.6	1.0	12%
DAP	1,086,729	44%	36.2	44.1	12.1	32%
RBESS	5,250,996	82%	97.0	47.9	1.5	30%
DTR	18,195,701	32%	252.1	176.1	3.9	53%
Exch	7,450,837	46%	37.4	41.3	1.6	17%
MSFS	29,345,085	33%	200.9	102.3	1.4	10%
BS	11,873,555	49%	149.8	166.2	3.7	67%

5.3 Evaluated Page-type Specifying Schemes

In our experiments, we evaluated six page-type specifying schemes, namely, **US**, **LFS**, **SBS+US**, **SBS+UBS**, **QDS+US**, and **QDS+UBS**. The details of these schemes are presented in Subsection 4.2. Specifically, the notations SBS+US, SBS+UBS, QDS+US, and QDS+UBS indicate paired-schemes, in which the second scheme is used as a complement to the first scheme. Take QDS+UBS 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 UBS scheme. The queue depth threshold of the QDS scheme is set to 10 requests in our experiments. For the SBS scheme, requests with only one sub-request involved are assigned the LSB pages while large writes are assigned by a complementary scheme, e.g., the US scheme for SBS+US. The HGS 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 WBS scheme is in fact reduced to the LFS scheme.

The conventional SSD design that allocates pages without considering the page-types is used as the baseline system for evaluating the performance of PA-SSD based on these page-type specifying schemes.

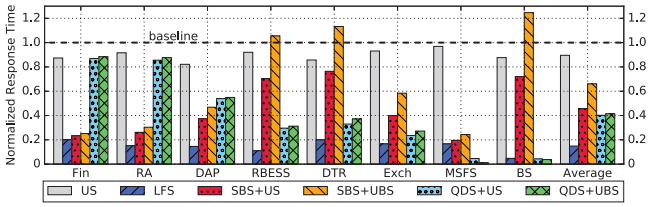
6 EXPERIMENTAL RESULTS

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

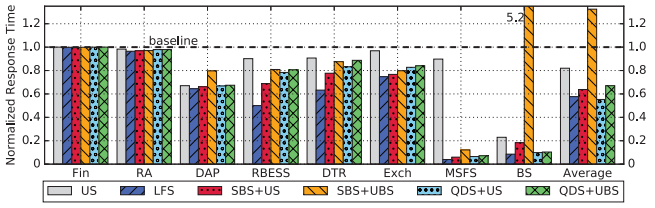
6.1 Average Response Time

In this subsection, we use the average response time, normalized to that the baseline, as a measure to evaluate the performance of PA-SSD with various page-type specifying schemes. The results on the average write response time and read response time are shown in Fig. 9. The workloads on the x-axis are sorted in their relative intensity increasing from light (left) to heavy (right).

The PA-SSD with US scheme reduces the write response time and read response time by 10% and 18% respectively on average. This



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



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

Figure 9: Performance evaluation of PA-SSD. Note that workloads on the x-axis are ordered in their relative intensity from light (left) to heavy (right).

performance improvement is attributed to a balanced proportion of *MSB dominated writes* and *LSB dominated writes* as PA-SSD with US effectively increases the *LSB dominated writes* to 34% while decreases the *MSB dominated writes* to 33%, as shown in Fig. 10

By prioritizing the LSB pages in allocation, LFS achieves significantly lower write and read response time than the baseline, by 85% and 42% respectively on average. The LFS scheme has the lowest write response time under most workloads except for the two heaviest workloads, i.e., MSFS and BS, where the LSB pages are rapidly depleted. We further analyze the reasons behind this in the next subsection.

The SBS+US scheme preferentially allocates LSB pages for small writes. Thus, for workloads with high proportions of small writes (low large write ratio), e.g., Fin and MSFS, SBS+US can greatly reduce the write response time. On average, SBS+US reduces write response time and read response time of the baseline by 54% and 36%, respectively. On the other hand, the SBS+UBS scheme tends to allocate CSB and MSB pages for large writes because the LSB pages have been depleted for serving small writes, resulting in longer average response time than the baseline in some cases.

The QDS+US 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., Fin and RA, the QDS+US scheme has an average write response time similar to the US scheme because the I/O queue in these workloads seldom exceeds the threshold. But for heavy workloads, especially for MSFS and BS, the QDS+US scheme achieves extremely low write and read response times. In short, QDS+US reduces write response time and read response time by 60% and 45% 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 QDS+UBS scheme achieves still lower write response time on the two heaviest workloads. Especially for MSFS, QDS+UBS reduces write response time of the baseline by 99%. Noticeably, the write response time of QDS+UBS is 6% higher than that of QDS+US under light workloads (i.e., from Fin to Exch), while its write response time is 56% lower than that of QDS+US under heavy workloads (i.e., MSFS and BS). This performance difference between QDS+US and QDS+UBS is explained in the next subsection.

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

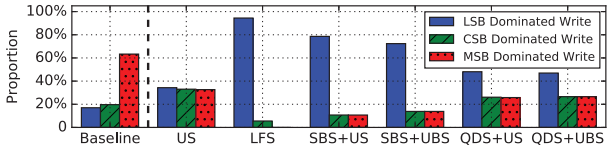


Figure 10: The distribution of *LSB/CSB/MSB dominated writes* under the DAP workload.

The performance improvement of PA-SSD over the baseline stems from its ability to effectively adjust the proportions of *LSB dominated writes*, *CSB dominated writes*, and *MSB dominated writes*. Fig. 10 illustrates the distribution of requests belonging to these three groups of write requests under the DAP workload for the baseline design and the main PA-SSD schemes. In the baseline design, the proportion of *MSB dominated 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 *MSB dominated writes* the lowest among the three groups (not higher than 1/3). By employing non-utilization-based page-type specifying schemes, e.g., LFS, SBS, and QDS, PA-SSD further improves write performance by promoting the proportion of *LSB dominated writes* and reducing the proportion of *MSB dominated writes*. Besides, using UBS instead of US as a complement to SBS and QDS always leads to more uniform distributions of the three groups.

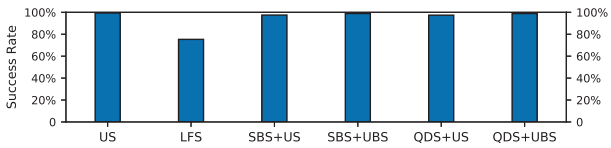
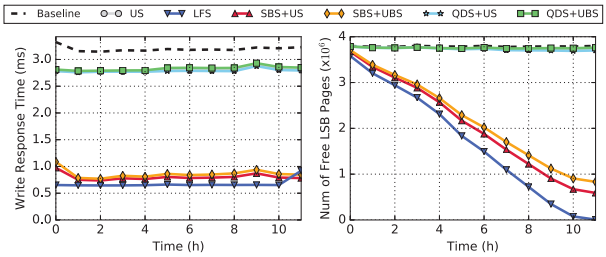
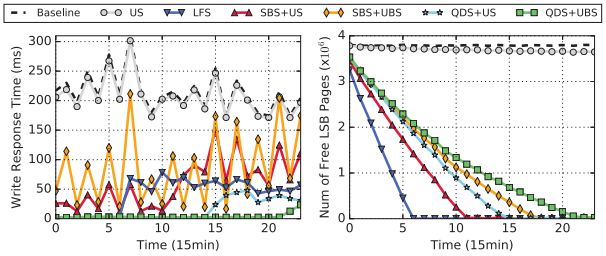


Figure 11: The success rate of PA-SSD on allocating specified-type of pages for sub-requests.

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. 11, the success rate is higher than 98% for five of the six evaluated schemes. The low success rate of PA-SSD with LFS is due to its greedy on using LSB pages, detailed in the next subsection.



(a) Fin



(b) MSFS

Figure 12: Write performance over time.

6.2 Write Performance Over time

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

We first analyze the write response time under Fin, which is depicted in the left diagram of Fig. 12(a). Because of the low I/O intensity and small ratio of large writes, even the baseline design can obtain a response time 21% higher than the average program latency in TLC flash (2.6ms in our simulated SSD). By allocating pages for sub-requests with pages of the specified page-type, US scheme reduces the write response time of the baseline by 13%. As the I/O queue under Fin seldom reaches the threshold of the QDS+US and QDS+UBS schemes, their performances are very close to that of US. On the contrary, the LFS 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. 12(a). 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 LFS scheme allocates the LSB pages is much faster than that of garbage collection to reclaim the LSB pages, resulting in significant degradation in write performance at the end of the simulation.

For MSFS, the LFS scheme achieves among the lowest response time during the first seven quarters (15mins/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. 12(b). On the contrary, both the QDS+US and QDS+UBS schemes maintain low write response times for a long period of time. Especially, QDS+UBS retains the low write response time for 22 quarters, which is three times that of the LFS scheme.

All of the PA-SSD schemes of LFS, QDS+US, and QDS+UBS improve the write performance by effectively leveraging the LSB pages. Among them, LFS, as the most radical scheme, can drastically reduce 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 QDS+US and QDS+UBS schemes are able to use LSB pages judiciously and efficiently. As a result, although QDS+US and QDS+UBS cannot achieve write response times as low as LFS on light workloads, they are capable of maintaining lower response times for longer periods of time under heavy workloads than the LFS scheme, achieving better overall performance. Additionally, by combining the UBS scheme that proactively reserves LSB pages during relative idle times, the QDS+UBS scheme performs even better than the QDS+US scheme under the extremely heavy workloads, e.g., MSFS and BS.

6.3 Performance in Long-term

As described above, the performance of LFS, QDS+US, and QDS+UBS 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, however, 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.

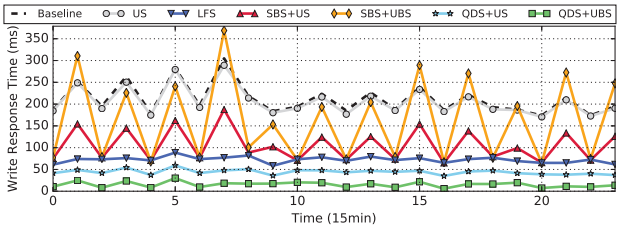


Figure 13: Performance in Long-term under MSFS workload.

We replay the MSFS workload multiple times continuously and observe that the free LSB pages in PA-SSD with the LFS, QDS+US, and QDS+UBS schemes are all used up before the third simulation. Therefore, we use the performance in the third simulation under MSFS to demonstrate the long-term performance of PA-SSD, which is illustrated in Fig. 13. As the figure shows, even when the free LSB pages are used up, PA-SSD with LFS, QDS+US, and QDS+UBS still outperforms the baseline significantly. Specifically, PA-SSD with the LFS, QDS+US, and QDS+UBS schemes achieve 66%, 79%, and 93% lower write response times than the baseline, respectively. Particularly, this long-term performance analysis suggests a great

potential of the PA-SSD with the QDS+UBS scheme in providing high and stable write performance in I/O-intensive environments.

7 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 allocates pages with the same type for sub-requests of any single user write request. Specifically, PA-SSD first assigns the page-types for user write requests according to seven proposed page-type specifying schemes, and then allocates pages for the corresponding sub-requests according to their assigned 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 SSD-Sim and evaluated its performance with various page-type specifying schemes. Our simulation results show that PA-SSD with the *QDS+UBS* scheme improves write and read performance by 2.4 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 research is sponsored by the National Basic Research 973 Program of China under Grant No. 2011CB302303, and the US NSF under Grant No. CCF-1704504 and CCF-1629625.

REFERENCES

[1] Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu. 2017. Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives. *Proc. IEEE* 105, 9 (Sept 2017), 1666–1704.

[2] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. 2013. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *IEEE 31st International Conference on Computer Design*. 123–130.

[3] Che-Wei Chang, Geng-You Chen, Yi-Jung Chen, Chia-Wei Yeh, Pei Yin Eng, Ana Cheung, and Chia-Lin Yang. 2017. Exploiting Write Heterogeneity of Morphable MLC/SLC SSDs in Datacenters with Service-Level Objectives. *IEEE Trans. Comput.* 66, 8 (Aug 2017), 1457–1463.

[4] Li-Pin Chang. 2010. A Hybrid Approach to NAND-Flash-Based Solid-State Disks. *IEEE Trans. Comput.* 59, 10 (Oct 2010), 1337–1349.

[5] Feng Chen, Tong Zhang, and Xiaodong Zhang. 2017. Software Support Inside and Outside Solid-State Devices for High Performance and High Efficiency. *Proc. IEEE* 105, 9 (Sept 2017), 1650–1665.

[6] Nima Elyasi, Mohammad Arjomand, Anand Sivasubramaniam, Mahmut T. Kandemir, Chita R. Das, and Myoungsoo Jung. 2017. Exploiting Intra-Request Slack to Improve SSD Performance. In *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*. 375–388.

[7] Laura M. Grupp, John D. Davis, and Steven Swanson. 2013. The harey tortoise: managing heterogeneous write performance in SSDs. In *proceedings of USENIX Annual Technical Conference*. 79–90.

[8] Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance Impact and Interplay of SSD Parallelism Through Advanced Commands, Allocation Strategy and Data Granularity. In *Proceedings of the International Conference on Supercomputing*. 96–107.

[9] Soojun Im and Dongkun Shin. 2010. *ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer.* Elsevier North-Holland, Inc. 641–653 pages.

[10] Myoungsoo Jung and Mahmut T. Kandemir. 2014. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *IEEE 20th International Symposium on High Performance Computer Architecture*. 524–535.

[11] Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim. 2009. FlexFS: a flexible flash file system for MLC NAND flash memory. In *proceedings of USENIX Annual Technical Conference*. 9–9.

[12] Yan Li, Cynthia Hsu, and Ken Oowada. 2014. Non-volatile Memory and Method with Improved First Pass Programming. US Patent. Patent number, US 8811091. (19 08 2014).

[13] Bo Mao and Suzhen Wu. 2015. Exploiting request characteristics and internal parallelism to improve SSD performance. In *Proceedings of the 33rd IEEE International Conference on Computer Design*. 447–450.

[14] Chihiro Matsui, Tomoaki Yamada, Yusuke Sugiyama, Yusuke Yamaga, and Ken Takeuchi. 2017. Tri-Hybrid SSD with storage class memory (SCM) and MLC/TLC NAND Flash Memories. In *Proceedings of the Flash Memory Summit*. 1–22.

[15] Muthukumar Murugan and David H. C. Du. 2012. Hybrot: Towards Improved Performance in Hybrid SLC-MLC Devices. In *Proceedings of the IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*. 481–484.

[16] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. 2016. Improving performance and lifetime of NAND storage systems using relaxed program sequence. In *proceedings of the 53nd ACM/EDAC/IEEE Design Automation Conference*. 1–6.

[17] SNIA IOTTA Repository. 2018. SNIA Block I/O Traces. http://iotta.snia.org/tracetypes/3. (2018).

[18] UMass Trace Repository. 2018. UMass Trace Repository Storage Traces. http://traces.cs.umass.edu/index.php/Storage/Storage. (2018).

[19] Deepak Sharma. 2014. System Design for mainstream TLC SSD. In *Proceedings of the Flash Memory Summit*. 1–20.

[20] Arash Tavakkol, Pooyan Mehrvarzy, Mohammad Arjomand, and Hamid Sarbazi-Azad. 2016. Performance Evaluation of Dynamic Page Allocation Strategies in SSDs. *ACM Trans. Model. Perform. Eval. Comput. Syst.* 1, 2, Article 7 (June 2016), 33 pages.

[21] Wei Wang, Wen Pan, Tao Xie, and Deng Zhou. 2016. How Many MLCs Should Impersonate SLCs to Optimize SSD Performance?. In *Proceedings of the Second International Symposium on Memory Systems*. 238–247.

[22] NVM Express Workgroup. 2017. NVM Express revision 1.3 specification. http://nvmexpress.org/. (2017).