

PcGC: A Parity-Check Garbage Collection for Boosting 3-D NAND Flash Performance

Shujie Pang, Yuhui Deng[✉], Genxiong Zhang, Yi Zhou[✉], Xiao Qin[✉], Zhaorui Wu[✉], and Jie Li

Abstract—Garbage collection or GC running in the controller of 3-D NAND flash-based solid-state disks—SSDs—plays a critical role in the performance of storage systems. SSD manufacturers have developed various GC solutions based on internal data movement or IDM to mitigate the impacts of GC on request latency. Due to the circuit characteristics of flash memory, the existing IDM-based GC strategies are restricted by page parity during data movement: odd pages must be migrated to odd pages, and even pages to even pages. When migrating two consecutive pages with the same parity, the free page between the two migrated pages will be wasted after the migration is complete. This ever-increasing page waste problem inevitably deteriorates the storage space utilization of flash memory, thereby degrading the overall performance of 3-D NAND flash-based SSDs. To address this issue, we propose a parity-check GC scheme called PcGC to revamp SSD performance by alleviating page waste during GC. We build a parity-check unit in PcGC to facilitate checking the parity of migrated valid pages and destination pages. According to the parity results offered by the parity-check unit, PcGC dynamically adjusts the migration order of valid pages during the course of GC. In doing so, PcGC fundamentally averts page waste caused by the page parity restriction, thereby enhancing 3-D NAND flash performance. We quantitatively evaluate the performance of PcGC in terms of wasted pages, storage utilization, GC counts, write amplification, and average response time. We compare PcGC against the two state-of-the-art schemes—Amphibian and Tiny-tail flash (TTflash). The experimental results derived from the nine real-world workload traces unfold that compared with Amphibian and TTflash: 1) PcGC curtails the number of wasted pages by up to 91.4% with an average of 53.75%; 2) cuts back the number of GC counts by up to 52.2% with an average of 11.9%; and 3) slashes

average write response time by up to 77.8% with an average of 13.0%.

Index Terms—3-D NAND flash, garbage collection, page waste, parity restriction.

I. INTRODUCTION

RECENT advances in NAND flash memory technology have fueled a revolutionary change in data storage space. Along with the evolution of flash memory from 2-D NAND structure to 3-D NAND structure, NAND flash-based solid-state disks—SSDs—have become the most popular storage devices adopted in a wide variety of fields, such as data centers, enterprise storage systems, personal computers, and mobile devices [1], [2], [3] thanks to NAND flashes' high performance, large capacity, and high scalability.

Though NAND flash-based SSDs significantly bolster the performance of storage systems, innate features out-of-place and erase-before-write turn into annoyance in NAND flash-based SSDs. The out-of-place trait entails that updated data cannot be directly overwritten on original addresses, while the erase-before-write characteristic specifies that flash blocks are obliged to erase old data before writing any new one. A flash translation layer (FTL) is employed on a NAND-flash SSD, which leverages a mapping table to perform logical-to-physical address translations, garbage collection (GC), and space management, thereby mitigating the out-of-place and erase-before-write issues. Unfortunately, FTL is still far from perfect for NAND flash-based SSDs. For example, to service a write request, FTL first allocates a handful of free pages for written data and then updates the mapping table. When it comes to an update request, however, such a process can cause serious storage waste. Specifically, FTL writes new data of an update request to free pages, leaving old data pages marked as invalid pages. These invalid pages remain for a long period of time until blocks containing these invalid pages are erased, thereby deteriorating space utilization of NAND flash memory.

GC plays a critical role in the performance of NAND flash-based SSDs. With the purpose of avoiding low space utilization caused by the innate features of NAND flash, FTL utilizes GC to migrate valid pages while recycling invalid ones. During GC, all valid pages in a victim block—the block with the most invalid pages—are migrated to other blocks, followed by erasing the victim block. Nevertheless, the downside is that GC may consume invaluable system resources to accomplish its task, which severely affects SSD performance. GC is divided into two categories—EDM-based GC and IDM-based

Manuscript received 6 November 2022; revised 9 February 2023 and 25 March 2023; accepted 26 May 2023. Date of publication 30 May 2023; date of current version 22 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62072214; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021B1515120048; and in part by the Open Project Program of Wuhan National Laboratory for Optoelectronics under Grant 2020WNL0KF006. The work of Xiao Qin was supported by the U.S. National Science Foundation under Grant IIS-1618669 and Grant OAC-1642133. This article was recommended by Associate Editor C. Zhuo. (Corresponding author: Yuhui Deng.)

Shujie Pang, Genxiong Zhang, Zhaorui Wu, and Jie Li are with the Department of Computer Science, Jinan University, Guangzhou 510632, China (e-mail: p_shujie@163.com; zgx787839110@163.com; diom_wu@163.com; lijiegxmd11@163.com).

Yuhui Deng is with the Department of Computer Science, Jinan University, Guangzhou 510632, China, and also with the Wuhan National Laboratory for Optoelectronics, Wuhan 430079, China (e-mail: tyhdeng@jnu.edu.cn).

Yi Zhou is with the TSYS School of Computer Science, Columbus State University, Columbus, GA 31097 USA (e-mail: zhou_yi@columbusstate.edu).

Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36830 USA (e-mail: xqin@auburn.edu).

Digital Object Identifier 10.1109/TCAD.2023.3281517

GC—according to the migration method. Since external data movement or EDM-GC occupies limited channel resources and leads to serious I/O latency, we mainly focus on internal data movement—IDM-GC, which is also known as *copy-back*. Because no external data migration operation occurs in IDM-GC, IDM-based GC operations tend to be faster than traditional EDM-based GC operations by up to 30% [4]. Meanwhile, IDM-based GC supports multiplane operations to process multiple GC requests in a chip synchronously.

The odd–even symmetry restriction in IDM-based GC imposes a performance bottleneck on modern NAND flash-based storage systems. Numerous SSD manufactures specify that if a valid page of a victim block is migrated to a free page in a free block, the two pages must share the same parity property. Put in plain words, an odd valid page must be shifted to an odd free page, and an even valid page to an even free page [5]. For instance, NAND Memory Toggle DDR1.0 interface [6] implemented by Samsung and Toshiba stipulates that IDM-based GC is required to be executed under the odd–even symmetry restriction. Other SSD manufacturers, such as Intel, Micron, and Hynix, developed the open NAND flash interface (ONFI) [7] establishing a similar odd–even requirement for GC. In addition, the 3-D Flash Memory Toggle DDR2.0 interface [8] continues implementing the IDM-based GC operations in 3-D NAND flash. Micron’s latest datasheets [9], [10], [11] show that current NAND flash memories are still limited by the odd–even symmetry restriction, and most of these datasheets are published in 2022. Specifically, in the parameter page data structure table of flash datasheets, bit 4 of bytes 6–7 controls the odd–even symmetry restriction switch of flash memories. If bit 4 is 0, the flash memories support the odd–even symmetry restriction. The odd–even limit inevitably gives rise to massive wasted pages. In particular, because the two consecutive valid pages with the same parity must be migrated to the two consecutive pages with the same parity in a free block during GC, the free page between the two migrated pages in the free block will be marked as invalid pages, resulting in one wasted page in the free block. This page waste phenomenon becomes even worse with the frequent triggering of GC, which has a huge impact on the storage space utilization of flash memory. It is worth noting that we refer to the odd–even symmetry restriction as *parity restriction* throughout this study.

Recently years have seen a growing trend toward expanding the capacity of SSDs by transitioning flash chips from 2-D NAND flash to 3-D NAND flash. Such a trend magnifies the impact of GC on SSD performance. A typical capacity increment example is that a block size is increased from 256 pages/block in 2-D planner NAND flash memory to 768 pages/block in 3-D NAND flash memory [12]. Unfortunately, given the block-size expansion, the number of pages migrated during each GC operation also goes up, thereby leading to additionally wasted pages. The degraded storage space utilization, in turn, triggers added GC operations, which dampens storage performance.

Motivated by the aforementioned observations and analysis, we propose a parity-check GC scheme coined PcGC to address the space-waste issue. We implement a parity-check unit in

PcGC that facilitates making decisions on page migration sequences amid GC. PcGC substantially stores odd and even pages alternately in free blocks, striving to curtail the number of wasted pages. Moreover, PcGC remarkably slashes the number of invalid pages in flash-based SSDs, the performance of which is enhanced. Meanwhile, PcGC, which is simple and well-suited for hardware implementations, can be heavily compatible with other SSD optimization strategies. Our contributions are summarized as follows.

- 1) We present two intriguing observations: a) the parity restriction of GC leads to considerable page waste and b) page waste significantly degrades the performance in the fields of 3-D NAND flash-based SSDs.
- 2) We propose a parity-check GC scheme—PcGC—to determine page migration sequences based on the parity of migrated pages: PcGC improves storage utilization and optimizes GC efficiency.
- 3) We conduct extensive experiments on nine real-world workloads to examine the effectiveness and efficiency of PcGC. The experimental results unveil that PcGC and the PcGC-based hybrid schemes not only extensively curb the number of wasted pages during GC but also substantially cuts back GC count and write-response time.

The remainder of this article is organized as follows. Section II summarizes the related work. The background is discussed in Section III. Section IV elaborates on the motivation of PcGC. Section V details the design of PcGC. Section VI is focused on the performance evaluation of PcGC. Finally, Section VII concludes this study.

II. RELATED WORK

As GC becomes a performance bottleneck for SSDs, many researchers have embarked on inspiring studies to alleviate the GC-induced performance degradation.

A GC operation consists of several read/write operations accompanied by a time-consuming erase operation, which yields a long tail latency problem. To alleviate the adverse impacts of erase operations on tail latency, schemes, such as erase suspension [12], preemption GC [13], and partial erase [2], were proposed to prioritize the user I/O requests. Wu and He [14] implemented the erase suspension techniques to suspend ongoing erase operations while servicing read requests. Unfortunately, Wu’s solution is incapable of resuming suspended erase operations in case of continuous read requests from hosts: starvation of erase and write operations is not a surprise. Lee et al. [15] designed a semi-preemptive GC to defer triggered GC operations. Regrettably, when a continuous I/O stream is processed, the semi-preemptive GC causes serious performance degradation. Unlike the above innovations that strive to prioritize the user I/O requests, our PcGC focuses on preventing GC. In particular, PcGC produces an optimized sequence of migrated pages to alleviate the problem of low storage utilization, thereby preventing an excessive number of GC operations.

GC plays a critical role in determining I/O response time because EDM-based GC occupies channel resources

and blocks I/O requests. Aiming to address this issue, Sha et al. [16] devised a low I/O GC scheduling strategy, which schedules GC operations in idle time slots for processing I/O requests, with the purpose of tackling the channel-blocking problem. Kang et al. [12] also adopted a related idea and introduced reinforcement learning techniques for I/O interval prediction, striving to resolve the conflict between GC-induced I/Os and host-induced I/Os. Mao et al. [17] proposed Amphibian that reduces the impact of channel occupation on I/O latency based on I/O characteristics. Although these methods relieve channel pressure to a certain extent, their effect is limited in the face of intensive requests.

Recently, much attention has been paid to improve the SSD efficiency of IDM-based GC. Compared with 2-D NAND, 3-D NAND flash-based SSDs have fewer channels, aggravating the channel-blocking problem. IDM-based GC draws researchers' attention thanks to its merit of supporting data migration in a plane to free channel resources. For example, Yan et al. [18] proposed Tiny-tail flash (TTflash) that exploits the fine-grained IDM to eliminate tiny-tail latency caused by GCs. TTflash compresses the impact of GC operations to a plane and eliminates read latency by parity-based RAIN. When processing write-intensive workloads, TTflash triggers a lot of extra read and write requests. Shahidi et al. [1] and Wang and Xie [19] adopted a similar idea to perform concurrent IDM-based GC operations within a chip, where plane-level utilization is optimized in NAND flash. Meanwhile, IDM-based GC does not flow to ECC of the SSD controller and thus cannot correct data errors. To solve this problem, Hong et al. [20] designed a frequency threshold mechanism to switch GC between the IDM mode and the EDM mode, and the solution well controls data error within a controllable range. Moreover, Chang et al. [21] developed a copy-back operation scheme to detect errors during IDM-based GC by embedding a hardware component in flash chips. Unfortunately, the aforementioned studies pay no attention to the limitation of 3-D NAND flash circuit characteristics on IDM-based GC. These circuit limitations cause IDM-based GC operations to trim the available storage space in flash-based SSDs. Unlike the existing solutions, our PcGC's focal point is to mitigate space waste incurred by the parity restriction of IDM-based GC, thereby radically reducing GCs due to insufficient space.

III. BACKGROUND

A. NAND Flash Organization

NAND flash-based SSDs consist of multiple flash chips and an SSD controller. Each flash chip is comprised of 2-D NAND or 3-D NAND dies; each flash die contains of multiple planes, each of which consists of thousands of blocks; each block is made up of dozens or hundreds of pages. Note that the unit of read and write operations is a page in NAND flash, whereas erase operations are performed on block granularity.

Fig. 1 shows a 2-D NAND flash and a 3D NAND flash, both of which are based on the odd-even structure. There are many studies and new chips based on the odd-even structure-based flash. For example, Micron's Naso et al. [22] developed a NAND flash with the odd-even structure. Macronix researchers studied

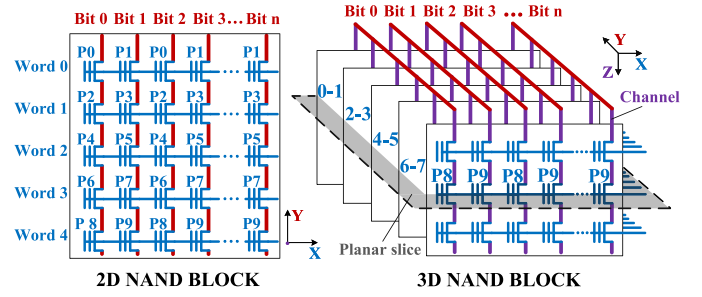


Fig. 1. 2-D/3-D NAND flash architecture.

layer-aware program-and-read schemes, which are optimized for an odd-even structure-based 3-D VG NAND flash [23]. Moreover, many optimization studies [24], [25], [26], [27] are also implemented on the basis of odd-even structure-based flash, the latest of which was published in 2021.

Although 2-D and 3-D NAND flash have the same logical structure, their micro-architectures are quite different in that 3-D NAND flash contains more pages per block than 2-D counterpart. Fig. 1 depicts the architectures of a planar 2-D NAND and a vertical channel 3-D NAND, which illustrates their obvious discrepancies in physical structure [28], [29]. More specifically, 2-D NAND is a planar structure that increases chip density by expanding horizontal areas. On the contrary, 3-D NAND stacks several planar slices to expand chip capacity. The structure of a horizontal slice in 3-D NAND is similar to 2-D NAND. In 2-D and 3-D NAND flash, each page is formed by a set of cells through a horizontal wordline (e.g., see pages P8 and P9 in Fig. 1), and bit lines connect cells of different wordlines in the same horizontal direction. On the other hand, in a vertical 3-D NAND, cells in different horizontal layers are connected by vertical channels. These architectural differences lead to the fact that the number of pages per block in a 3-D NAND flash is much higher than that in a 2-D NAND flash. In particular, two main reasons contribute to 3-D NAND flash's larger capacity: 1) unlike 2-D NAND, 3-D NAND stacks planar NANDs to inflate SSD capacity and 2) the block-related control circuit in 3-D NAND is located on the top layer, which controls all cell signals in its underlying layers. And as more layers are stacked, the control circuitry also increases. Unfortunately, the horizontal area that can be controlled by such a block-related control circuit is limited, which reduces the number of blocks with an increasing number of pages per block [2].

NAND flash usually utilizes *sequential write* to write data into cells: bitlines write data sequentially from top to bottom (e.g., from P0 to P9 in the example shown in Fig. 1). Moreover, because of different addresses of bitlines, cells on the same wordline are usually divided into two groups: 1) an odd group and 2) an even group [27], forming odd pages and even pages, respectively. For example, page P8 and page P9 in Fig. 1 are constructed by an even group and an odd group in wordline 4, respectively. Sequential write, together with the parity restriction in NAND flash, induce storage waste during GC.

B. Garbage Collection

NAND flash prompts GC when there is insufficient free space in a flash chip. Each GC operation encompasses four steps:

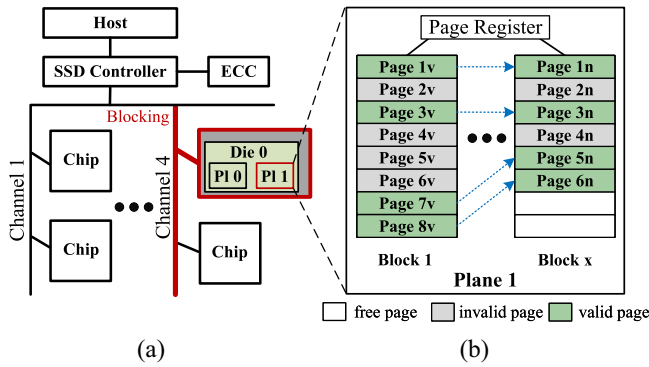


Fig. 2. Two GC operations overview. (a) External data move. (b) Internal data move.

1) select a victim block with the most invalid pages; 2) read the content of valid pages from the victim block; 3) write the content to a free block; and 4) erase victim blocks that contain no valid pages. After steps 2 and 3, the migrated valid pages in the victim book are marked as invalid. Obviously, the number of valid pages determines the overhead of a GC operation.

Recall that GC is usually classified into EDM-based and IDM-based GC according to the data migration strategy. The EDM-based migration methods occupy SSD channels and generate considerable I/O latency. Specifically, when performing an EDM-based GC operation, the data of a valid page is read into the page register and transferred to the SSD controller through a channel. Then, the ECC-checked data is written back to a new free page. During this process, the involved channel cannot be released until the victim block is erased. For example, Fig. 2(a) shows that if plane 1 is undergoing a GC operation, all chips in channel 4 will be blocked until the GC operation on plane 1 is complete.

IDM-based GC, on the other hand, is positioned to accelerate data migration, thereby shortening I/O latency. IDM-based GC does not pass the data of migrated pages through the SSD controller: channel resources are discharged and I/O latency is slashed. Fig. 2(b) illustrates an IDM-based GC example—if an IDM-based GC operation is performed on plane 1, valid pages in block 1 will be read into the page register and directly written to block x. Then, block 1 is erased after all valid pages are shipped to block x. Since these operations are carried out inside plane 1, there is no blocking on I/O requests to other storage modules. Because IDM-based GC is incapable of error detection and correction via ECC, many studies chose to avoid data errors by controlling IDM frequency or integrating additional error-detecting hardware. This practice promotes SSD manufacturers to adopt IDM-based GC in flash memory.

To give insight into the advantages of IDM, we investigated a performance comparison between EDM-based and IDM-based SSDs. Fig. 3 depicts the average response time of the IDM-based and EDM-based SSDs for processing read/write requests issued by the host. In order to intuitively and clearly show the experimental results, we normalize the experimental results of the IDM-based SSD to the level of the EDM-based SSD. We can observe from Fig. 3 that compared with the EDM-based SSD, the IDM-based SSD has lower read and write response time, which demonstrates its outstanding

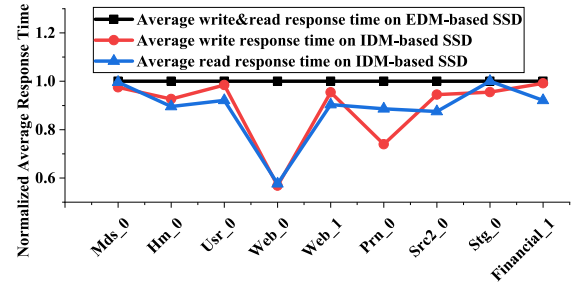


Fig. 3. Latency comparison between EDM-based and IDM-based SSDs.

SSD performance. IDM-based GC frees up channel resources and compresses the impact of GC into a plane. Therefore, IDM-based SSDs can effectively reduce I/O latency, thereby improving the overall SSD performance.

C. Parity Restriction

Presently, flash memory manufacturers are mainly divided into two interface camps: 1) ONFI (Micron, SK Hynix, Western Digital, et al.) and 2) Toggle (Samsung and Toshiba). ONFI and Toggle clearly stipulate that a LUN contains at least one-page register and a flash array, and the number of page registers depends on the number of plane operations supported for the LUN, which means that a plane corresponds to a page register. Since IDM (Copyback) is performed within a plane, only a one-page register is used. Moreover, ONFI explicitly stipulates that “Copyback uses a single page register for the read and program operation [7].” Therefore, we can conclude that odd and even pages share a page register. However, ONFI and Toggle interface propose parity restriction on IDM-based GC: the content of odd pages in a victim block must be migrated to odd pages in a free block. Likewise, even pages are diverted to even pages. Specifically, IDM operations are supported in the 3-D NAND interface Toggle 2.0, but there is no explanation and definition of IDM [8]. Its definition and explanation are completed in Toggle 1.0 [6], which stipulates the parity restriction. Likewise, ONFI also supports the restriction. While ONFI does not explain the details of the restriction, Toggle indicates that “the least significant bits (LSBs) of page address shall be the same between source and destination pages. In other words, the page of even page address cannot be copied to the page of odd page address, and the page of odd page address cannot be copied to the page of even page address as well [6].”

Unfortunately, when this parity restriction and the sequential write restriction exist at the same time, page waste becomes inevitable. Again, let us take Fig. 2(b) as an example of IDM-based GC, assuming block 1 is a victim, and block x is a free block; pages 1v, 3v, 7v, and 8v are valid pages, and other pages are invalid pages of block 1. After GC, the contents of valid pages 1v, 3v, 7v, and 8v are shifted to free pages 1n, 3n, 5n, and 6n, respectively. At this point, because of the parity restriction of IDM-based GC, pages 2n and 4n are marked as invalid—two wasted pages are expected. As NAND flash continues provoking GCs, the phenomenon of page waste becomes severe. In the worst case, a GC operation can waste

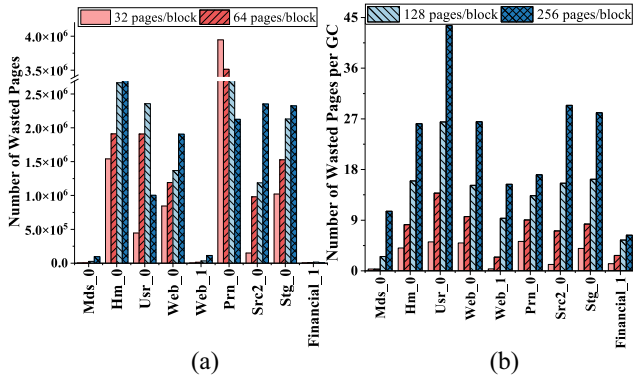


Fig. 4. Number of wasted pages for different block sizes. (a) Amount of wasted pages. (b) Wasted pages per GC.

half of the pages in a block. ONFI and Toggle do not mention that the parity restriction of IDM-based GC is affected by different bitline architectures.

IV. MOTIVATION

As the capacity of a single flash chip expands, 3-D NAND flash-based SSDs have reduced the number of channels with low channel-level parallelism. For this reason, IDM-based GC, as a channel-free method, is frequently used for collecting invalid space in flash memory. However, the parity restriction results in remarkable waste of storage space. To gain insights into the impacts of wasted pages, in this section, we first examine the impact of pages per block on page waste and then demonstrate the influence of page waste on performance.

A. Impact of Pages Per Block on Page Waste

The number of pages per block escalates as 3-D NAND density expands, increasing the number of valid page migrations during GC. More often than not, this phenomenon is referred to as the “Big Block” problem [30], which exacerbates wasted space due to the parity constraint in IDM-based GC.

We conduct four experiments to navigate the impact of block size on the number of wasted pages. To prevent the interference of SSD capacity in GC frequency, we carry out the experiments under four iso-capacity SSD configurations (i.e., the number of pages per block and the number of chips), namely, (32, 16), (64, 8), (128, 4), and (256, 2). For fair comparison purpose, we fix the number of channels to two across these four configurations, such that the channel parallelism of the four configurations is at the same level. We initiate this group of experiments with nine storage workloads on the SSDsim [5] simulator, and the characteristics of these workloads are tabulated in Table II.

Fig. 4(a) depicts the number of wasted pages with respect to block sizes under the nine workload conditions. It reveals that the number of wasted pages rises as the number of pages per block soars in most workloads. Except workloads *Prn_0* and *Usr_0*, the number of wasted pages reaches a peak when each block has 256 pages. The different trend on *Prn_0* and *Usr_0* is attributed by the fact that compared with other workloads, the GC counts of a small-block flash are much higher than that of a large-block flash on *Prn_0* and *Usr_0*. Specifically,

the total wasted pages = GC counts × wasted pages per GC. Fig. 4(b) shows that the average number of wasted pages per GC has an ascending trend as the number of pages per block grows. However, in Fig. 4(a), as the number of pages per block increases, the total number of wasted pages on *Prn_0* and *Usr_0* decreases. Thus, on the two workloads, the GC counts exhibit a downward trend, and the GC counts of a small-block flash significantly exceed that of a large-block flash. In the following, we explain the reasons for this phenomenon.

Table II reveals that the actual WS of *Prn_0* and *Usr_0* is the largest on the nine workloads (please refer to the description in Section VI-A). Unfortunately, a larger actual WS represents a larger free space requirement and generates more GC operations. Besides, compared with a large-block flash, a small-block flash usually requires more GC operations to generate sufficient free space. For example, assume that a large-block flash chip is 256 pages per block, and a small-block flash chip is 64 pages per block. When write requests with 256 pages are, respectively, written to the two flash chips, the large-block flash chip generates 256 free pages by a single GC operation for serving these requests, while the small-block flash chip requires four GC operations. The difference between the GC counts of the large-block flash chip and the small-block flash chip is 3. When the write request size increases to 384 pages, the small-block flash chip and the large-block flash chip generate six GCs and two GCs, respectively. At this point, the difference between the GC counts is 4. It can be seen from the example that compared with the small written size, the difference between the GC counts will increase when serving larger write requests. As mentioned above, since *Prn_0* and *Usr_0* have the largest actual WS among the nine workloads, there is also the largest difference in GC counts between a large-block and a small-block flash. Compared with other workloads, the GC counts of a small-block flash are much higher than that of a large-block flash on *Prn_0* and *Usr_0*. Since page waste is generated by GC operations, the more GC operations, the more page waste. Therefore, in the case of an extreme difference in GC counts, *Prn_0* and *Usr_0* may have different trends.

Furthermore, although the GC counts of small-block flash will be higher than that of large-block flash when handling large write requests, flash memory has the upper bound of total GCs (TGCs). When flash chips reach the upper bound of TGC, the total number of wasted pages is determined by the wasted pages per GC. The detailed reason is explained as follows. First, we must clarify that the upper bound of TGC in flash memory is related to the number of blocks. Especially, the upper bound of TGC = the number of blocks × the upper bound of GCs per block. In a 3-D NAND flash, pages per block are increased by stacking more layers. However, since the number of blocks per plane is limited by the physical structure, planes with similar physical size have a similar number of blocks in small-block and large-block flash [29], [31]. Second, due to the limited number of erases of flash cells, the number of GC operations performed by a block is limited. Consequently, large and small blocks with the same type of cells have the same upper bound of GCs per block. Specifically, when a flash chip reaches the upper bound of

TGC, the total number of wasted pages is: the number of blocks $\times$ the upper bound of GCs per block $\times$ the number of wasted pages per GC. As mentioned above, the number of blocks in large-block and small-block flash is similar, and the upper bound of GCs per block is the same. At this point, wasted pages per GC determines the total wasted pages of flash memory. Therefore, we discussed wasted pages per GC with different block sizes in Fig. 4(b), which effectively evaluates the page waste in old flash memory that reaches the upper bound of TGCs.

Fig. 4(b) depicts the average number of wasted pages per GC with respect to block sizes on the nine workloads. It shows that the average number of wasted pages per GC has an ascending trend as the number of pages per block grows. And the number of wasted pages reaches a peak when the number of pages per block increases to 256. For example, on workload *Usr_0*, the number of wasted pages per GC climbs from three pages to a maximum of forty-three pages. On average, each GC operation wastes 10% of the storage space of a block. Such a page waste phenomenon will exacerbate with subsequent triggered GC operations if the page distribution is not adjusted in blocks of flash chips.

B. Impacts of Wasted Pages on SSD Performance

Page waste inevitably leads to SSD performance degradation: wasted pages trim free storage space in flash chips, which in turn frequently provokes GC operations, thereby seriously harming the reliability and lessening the performance of 3-D NAND flash-based SSDs.

To verify that it is achievable to maintain high SSD performance by alleviating page waste, we investigate the impact of wasted pages on average write response time and GC counts under the nine workloads. Average write response time and GC counts are two popular metrics that quantitatively assess SSD performance (see Section VI-A for the detailed performance metrics). We undertake this group of experiments on the nine workloads with respect to two cases with parity restriction and without parity restriction. No parity restriction means no wasted pages. To achieve fair comparisons, we normalize the experimental results of different cases to the level of the parity restriction case. Please refer to Section VI for the detailed experimental settings. The results of this group of experiments are plotted in Fig. 5, where low response time and few GC counts indicate high SSD performance. Fig. 5 shows that the absence of parity restriction immensely reduces response time and GC counts. These experimental results confirm that page waste degrades the performance and lifetime of SSDs.

V. PCGC DESIGN

We start this section by presenting the overview of our PcGC scheme. Then, we introduce the implementation details of PcGC, which embraces a parity-check unit and a data migration module.

A. System Overview of PcGC

Due to the parity restriction of IDM-based GC, 3-D NAND flash-based SSDs may suffer from excessive physical storage

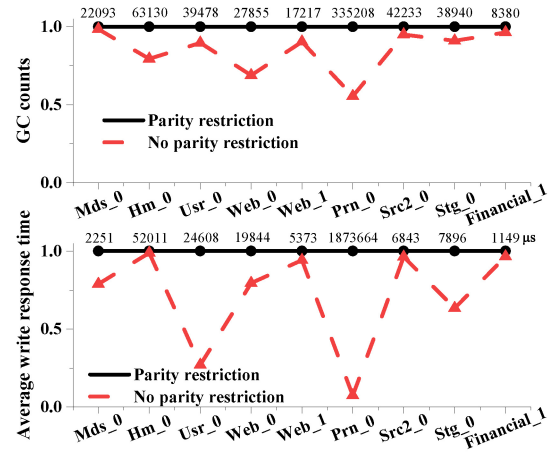


Fig. 5. Correlation among wasted pages, GC counts, and average write response time (normalized).

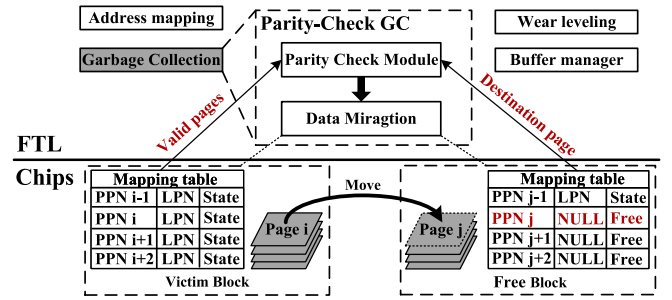


Fig. 6. Overview of PcGC for NAND flash-based SSDs.

space waste. To mitigate this concern, we propose the parity-check GC technique—PcGC—that is anchored on IDM-based GC. PcGC differentiates itself from the existing GC schemes by restraining the waste of physical pages incurred by parity restriction.

Fig. 6 depicts the overall system design of our proposed PcGC, which is deployed in the flash transaction layer (FTL) of SSDs. The overarching goal of PcGC is to eliminate wasted pages through proper data migration scheduling during the course of GC. To this end, PcGC leverages the parity of pages to determine the migration order of valid pages. At the heart of PcGC are two modules, namely, a parity-check unit and a data migration module. The parity-check unit is responsible for checking the parity of pages (i.e., even or odd) in a victim block and a free block during PcGC. According to the parity outputted by the parity-check unit, the data migration module adjusts the migration order of valid pages to avert page waste.

PcGC strives to optimize storage space utilization during the course of its GC. PcGC is normally invoked in the following two situations: 1) the free space in a plane is insufficient to service subsequent write requests and 2) a plane performs wear-leveling operations on blocks. When one of the above two conditions is held, a plane will perform a PcGC operation and suspend services correspond to read/write requests. PcGC performs its GC operations from the perspective of the two cases. In particular, if a victim block in a plane has no valid pages, PcGC will directly erase the block; otherwise, PcGC will invoke its data migration operation. That is, valid pages in

the victim block will be migrated into a free block in the same plane. Throughout the migration process, PcGC judges the parity of pages in the victim and the free block to dynamically adjust the migration order of the migrated valid pages.

B. PcGC Design

Unlike the existing IDM-based GC solutions, PcGC is anchored on directly optimizing the data migration process, thereby knocking out the waste of storage space. More pro-saically, PcGC manages the migration of valid pages during data migration in accordance with parity restriction. At the heart of PcGC is the parity-check unit. In what follows, we explain the role of the parity-check unit, followed by the detail of PcGC.

The Parity-Check Unit: In the data migration process of PcGC, the parity-check unit facilitates migrating valid pages in victim blocks to free pages in free blocks in a storage-efficient manner. Flash memory chips need to maintain wear-leveling. That is, the number of erase operations per block is balanced. According to the wear-leveling principle, when a victim block activates a PcGC operation, the parity-check unit first selects a free block on the same plane. Then, the unit locates the first free page after the last valid page in this free block as the *destination page* for writing migration data. In particular, the parity-check unit involves the following three steps to determine a migration order of valid pages to avoid wasted pages: 1) reading the parity of the destination page in the selected free block; 2) loading the parity of valid pages in a victim block; and 3) comparing the parity of the destination page with the parity of all the valid pages until a valid page with the same parity as the destination page is found; and this valid page is marked as a *parity symmetrical page*.

The traditional GC methods read valid pages in a victim block sequentially according to page order. Then, the valid pages are written to a free block in the same order. As a result, these GC methods lead to page waste due to ignoring the parity restriction. Fortunately, there exist no restrictions on read order in SSDs. Leveraging this feature, PcGC successfully mitigates the parity restriction issue that afflicts the traditional GC methods and optimizes the data migration process in GC, thereby reducing the number of wasted pages.

Algorithm 1 outlines the pseudocode of the GC process in PcGC. When the insufficiency of storage space condition or the wear-leveling principle provokes a GC operation, PcGC searches a victim block on the plane that triggers this PcGC operation. A victim block is the one with the most invalid pages. If the victim block contains valid pages, PcGC migrates the valid pages one by one from the victim block to a selected free block (see Line 4 in Algorithm 1). More specifically, PcGC uses *Get_destination_page()* to obtain the address of the destination page and employs *Get_ppn()* to calculate the physical page number (PPN) of the page. According to the result of $PPN\%2$, the parity-check unit determines the parity of the destination page in the selected free block and employs the *Get_parity_page()* function to find the *parity symmetrical pages* in the victim block (see Lines 6–11 in Algorithm 1). If there exists a parity symmetrical page in the victim block, PcGC directly writes the content of the parity symmetrical

Algorithm 1 PcGC Scheme

Input: Victim block $Block_1$
Output: Free block $Block_1$ (eased)

```

1: procedure PCGC DATA MIGRATION OPERATION
2:   if  $Block_1$  has no free pages then
3:     /* $Block_1$  triggers GC*/
4:     while  $Page\_first \neq NULL$  do
5:       /* $Page\_first$  is the first valid page*/
6:        $Page_b = Get\_destination\_page(Block_2)$ 
7:       /*Find the destination page and return*/
8:        $PPN = Get\_ppn(Page_b)$ 
9:       /*Get the physical page number*/
10:       $Page = Get\_parity\_page(Block_1, PPN\%2)$ 
11:      /*Find the parity symmetrical page in  $Block_1$ */
12:      if  $Page \neq NULL$  then
13:         $Move\_data(Page, Page_b)$ 
14:         $Page = invalid\ page$ 
15:      else if  $Page == NULL$  then
16:        /*No parity symmetrical page*/
17:         $Move\_data(Page\_first, Page_b + 1)$ 
18:         $Page\_first = invalid\ page$ 
19:      end if
20:      if  $Page\_first == invalid\ page$  then
21:         $Page\_first = Get\_frist\_page(Block_1)$ 
22:        /*Find a new first valid page*/
23:      end if
24:       $Update(Block_1, Block_2)$ 
25:      /*Update the page state in the blocks*/
26:    end while
27:     $Erase(Block_1)$ 
28:  end if
29: end procedure

```

page to the destination page (see Lines 12–14 in Algorithm 1). Otherwise, if PcGC finds no parity symmetrical pages in the victim block, PcGC moves the first valid page in the victim block to the free block (see Lines 15–18 in Algorithm 1). In particular, if the first valid page is migrated, PcGC will search for a new first valid page (see Lines 20–22 in Algorithm 1). Then, PcGC updates the information of migrated pages in both the victim block and the free block. PcGC repeats the above operations until no valid pages can be found in the victim block. After that, PcGC erases the victim block.

In the following example, we demonstrate how our PcGC effectively eliminates wasted pages. We assume that the victim block has five valid pages, 1v, 3v, 7v, 8v, and 10v, to be shipped to the free block (see Fig. 7). In a conventional GC scheme, the data migration order for these valid pages is ① to ⑤. That is, the five valid pages in the victim block are read and written to the free block sequentially from top to bottom. Consequently, there exist three wasted pages, namely, 2n, 4n, and 7n, in the free block after the GC operation. Using the most common conventional GC can easily and significantly reveal the impact of the parity restriction on GC. Many studies usually employ this conventional GC for comparison and description [32], [33]. We also admit that many GC methods have been proposed to solve the GC problems (please see related works in Section II as well). However, these methods ignore the parity restriction, which suffers from the same problems as the conventional GC does. Therefore, in order to facilitate readers' comprehension, we adopt the conventional GC to describe the advantages of PcGC.

Unlike the traditional GC schemes, PcGC takes an optimized page migration order to avoid page waste amid PcGC GC (see Fig. 7). More specifically, PcGC first opts for page

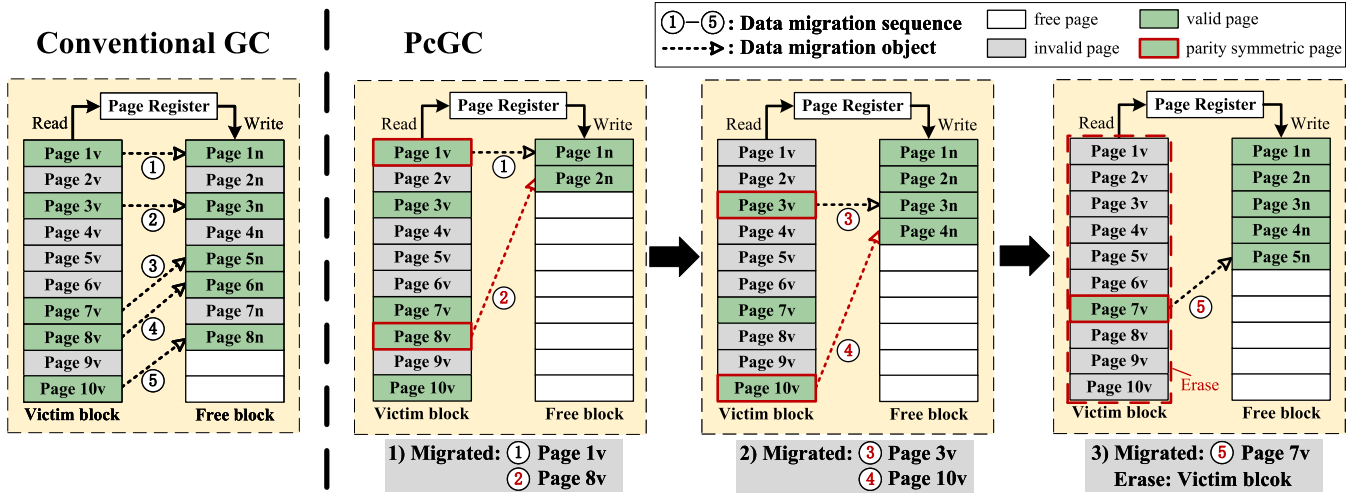


Fig. 7. PcGC scheme workflow. a) Migrated: ① Page 1v ② Page 8v, b) Migrated: ③ Page 3v ④ Page 10v, c) Migrated: ⑤ Page 7v Erase: Victim block

In the free block as the destination free page. Next, PcGC inspects the parity of valid pages in the victim block, aiming to locate a parity symmetrical page corresponding to page 1n. Since page 1v has the same parity as page 1n, page 1v is marked as a parity symmetrical page corresponding to page 1n. Put in another way, PcGC migrates the data of page 1v to page 1n. Then, PcGC elects page 2n as the destination free page, and picks up page 8v as the symmetrical page of page 2n for migration. PcGC chooses page 8v instead of pages 3v and 7v because the parity of pages 3v and 7v differs from the parity of page 2n. After moving the data of page 8v to page 2n, the destination free page becomes page 3n with its corresponding parity symmetrical page 3v, then page 4n with its corresponding parity symmetrical page 10v, and finally page 5n with its corresponding parity symmetrical page 7v. Finally, upon the completion of the migration of these five valid pages, PcGC erases the victim block. Fig. 7 unfolds that with PcGC in place, the flash memory produces a new free block and a block without any wasted pages. Moreover, even in this single-block migration example, PcGC saves a storage space of three pages compared to the traditional GC solution. In a nutshell, PcGC, being aware of page parity, fundamentally avoids page waste by adjusting the page-migration order.

C. Discussion

1) Multibit NAND Flash: To facilitate the description of the parity restriction, we adopted SLC NAND to introduce PcGC in this article. However, we have to argue that the parity restriction introduced by ONFI and Toggle are not limited to specific NAND types. Especially, Toggle datasheets [6], [8] apply and explain IDM and the parity restriction, which are the datasheets for the MLC and TLC NAND flash memory, respectively. As mentioned in Section III, the parity restriction explained in Toggle is “the LSBs of page address shall be the same between source and destination pages.” Specifically, an MLC wordline is divided into an odd LSB page, an even LSB page, an odd most significant bit (MSB) page, and an even MSB page [24]. Since the page address per wordline in

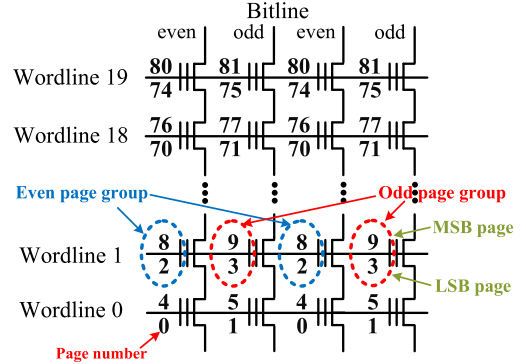


Fig. 8. MLC NAND flash even/odd bit-line structure.

multibit NAND (MLC, TLC, QLC, et al.) contains the LSB, multibit NAND is still affected by the parity restriction.

Furthermore, Multibit NAND flash stores multiple bits in a single cell and increases the number of physical pages per wordline. Specifically, MLC, TLC, and QLC flash can store two, three, and four bits per cell, respectively. They also form multiple types of pages with different types of bits. Taking MLC as an example, MLC NAND flash designs two bits per wordline into two types of pages: 1) LSB page and 2) MSB page. Since multibit flash stipulates that pages per wordline must be programmed sequentially from LSB to MSB pages, this fact may change the program sequence of physical pages in a block. Fortunately, such a change does not affect the data migration optimizations during GC achieved by PcGC. Let us still take MLC NAND flash as an example. Fig. 8 shows the page distribution structure of an MLC NAND flash from [27]. This structure is also adopted in the experiments of [24]. In Fig. 8, wordline 1 includes four physical pages: 1) LSB even page 2; 2) LSB odd page 3; 3) MSB even page 8; and 4) MSB odd page 9, where the page number represents the program order of the physical pages. It is worth noting that since multibit 3-D NAND flash adopts a new program allocation to reduce program disturbance, the program order of LSB and MSB pages on the same wordline is discontinuous [34], [35]. When data is written to the MLC flash, the pages in the flash are

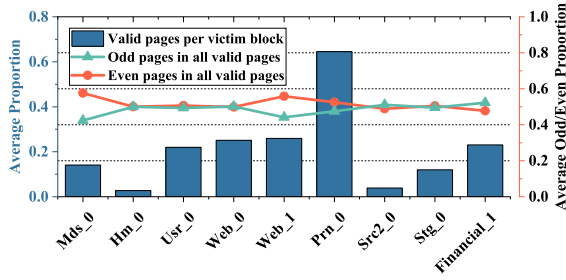


Fig. 9. Distribution of valid pages per victim block.

programmed sequentially from page 0 to n . In wordline 1, LSB pages 2 and 3 are programmed first. Then, after pages 4–7 on other wordlines are programmed, MSB pages 8 and 9 will be programmed. Since 1) pages in multibit flash are still alternately programmed with odd and even pages and 2) our PcGC determines the migrated order by checking the parity of the destination pages and valid pages, PcGC can alleviate space waste of multibit NAND flash.

2) *Data Locality*: PcGC changes the data locality (the continuity of data) for saving storage space, whereas this changed data location will not affect SSD performance. The main reason is that the migrated valid data still stay in the same block. In particular, the basic read/write unit of flash memory is a page, and the erase unit is a block, thus changing the location of data pages in a block will not affect the performance of read/write and erase operations. Moreover, data locality usually affects SSD performance from two aspects: 1) GC and 2) read/write parallelism. When contiguous logical pages are stored in consecutive physical pages, page fragmentation can be effectively reduced, thereby improving GC performance. At this point, PcGC only adjusts the physical location of pages in a block and does not affect the number of invalid and migrated pages in the block. Second, to improve the parallelism of read and write operations, some allocation schemes place contiguous logical pages in different parallel units, such as channels, chips, and planes. Under these allocation schemes, the continuity between physical pages within a block is negligible. Therefore, the location change caused by PcGC does not degrade SSD performance.

3) *Valid Page Distribution*: Since the distribution of valid pages may affect the PcGC performance, we discussed the distribution and the number of valid pages in a victim block. The bar charts in Fig. 9 illustrates the average proportion of valid pages per victim block. In Fig. 9, the average proportion of valid pages in each victim block is about 20% on all workloads except *Prn_0*. Under workload *Prn_0*, the average proportion reaches 64.4%. However, since our PcGC reduces page waste by adjusting the migration order of odd and even pages, the number of valid pages does not affect PcGC. Meanwhile, according to the physical location, valid pages are divided into odd valid pages and even valid pages. The number of odd and even valid pages may affect the performance improvement of PcGC. Therefore, we also experimentally discuss the distribution of odd and even valid pages. The curves in Fig. 9 shows the average proportion of odd and even valid pages on nine traces. We can observe from Fig. 9 that the number of odd

and even valid pages in all valid pages is similar. Even in the worst case, odd valid pages are only 15% fewer than even valid pages. Undoubtedly, this relatively balanced parity is friendly to our PcGC. PcGC can complete the interleaved migration of odd and even pages under the relatively balanced odd/even ratio, thereby alleviating page waste.

4) *Overhead of PcGC*: We have discussed the overhead of PcGC in terms of space and time overhead. First, the space overhead of PcGC is negligible. To detect the parity of pages, PcGC obtains the physical address of valid pages in a victim block and the destination page in a free block, then checks the parity of the pages. Fortunately, both the page address information and the PPN are existing information in SSDs. Since the parity of a page can be easily detected by the page's PPN, PcGC does not incur additional space overhead.

Second, the time complexity of PcGC is $O(n)$, where n denotes the number of valid pages from the first valid page to the parity symmetrical page in a victim block. However, the distribution of odd and even valid pages usually affects the value of n . Taking Fig. 7 as an example, since the odd and even valid pages are, respectively, distributed at both ends of the victim block, finding an even valid page requires to traverse all the odd valid pages. Specifically, to find the parity symmetrical page $10v$ in migration (4), valid pages $7v$ and $10v$ ($n = 2$) must be traversed sequentially. To address this issue, we employ two search pointers in PcGC, which point to odd and even valid pages, respectively. PcGC uses the two pointers in turn to migrate odd and even valid pages. As a result, the value of n is the number of valid pages from the last migrated odd/even page to the parity symmetrical page. For example, PcGC only needs to traverse valid page $10v$ in migration (4) ($n=1$). Furthermore, as shown in Fig. 9, the proportion of valid pages in a victim block is around 20% on most traces. Taking a flash memory with 512 pages per block as an example, there are only about 100 valid pages per victim block on average. Moreover, as the data migration progresses, the number of valid pages and the value of n gradually decreases. Therefore, n is usually small, and the time overhead of PcGC is limited.

VI. EVALUATION

A. Experimental Setups

To facilitate accurate performance analysis at the storage level, we choose SSDsim [5] as our experimental platform to implement our PcGC as well as the competitors. SSDsim, a trace-driven SSD simulator, seamlessly integrates well-defined structures, such as buffer management, a request allocation layer, an FTL layer, and a hardware module layer. In our evaluation, most of the SSD parameters are adopted from TTFash [18] published in FAST, which is also one of our comparison schemes. Thus, we believe that the configuration of TTFash has been shown to be effective for performance evaluation. In addition, we employ nine traces with different sizes and characteristics to evaluate our PcGC. Since the number of write requests leads to a limited number of GC operations, and some traces only contain a small amount of written data, there is a lack of sufficient GC operations to evaluate

TABLE I
SIMULATION PARAMETERS [18]

Size	Latencies
<i>SSD Capacity</i> 16G	<i>Page program</i> 800 μ s
<i>Channels</i> 8	<i>Page read</i> 40 μ s
<i>Chips</i> 16	<i>Block erase</i> 2000 μ s
<i>Blocks/plane</i> 4096	FTL
<i>Pages/block</i> 64	Over provisioning ratio 0.1
<i>Page size</i> 4KB	Cleaning policy Greedy

the performance of different schemes. Thus, we believe that a small-capacity SSD will facilitate generating enough GC operations, which can effectively improve the evaluation accuracy. Therefore, we reduce the number of chips adopted from TTflash and set the size of SSD flash memory to 16G. Furthermore, to ensure the effectiveness of the comparison scheme [18], we downloaded the source code of the comparison scheme which requires multiple channels and chips to demonstrate its performance advantages. Because its source code files indicate that some parameters (such as the number of channels) should not be changed, we adopt the structure of 8 channels and 16 chips. In a nutshell, we adopted the experimental settings in this article to accurately evaluate the performance of different schemes when dealing with a large number of GCs. Table I summarizes the detailed configurations of the simulation in this study.

Comparison of Methods: We compare PcGC with two alternative solutions TTflash [18] and Amphibian [17]. Meanwhile, we adopt the basic GC scheme in SSDsim as the baseline solution. We download the source code of TTflash—an open project based on SSDsim; we implemented IDM-based Amphibian on SSDsim. Furthermore, because our PcGC’s focus on page waste during data migration can friendly coexist with most GC-optimization techniques, we investigate the performance of hybrid solutions TTflash-PC and Amphibian-PC by integrating PcGC with TTflash and Amphibian, respectively. TTflash and Amphibian are highlighted as follows.

- 1) *TTflash*: TTflash aims to abate the impacts of GC by leveraging internal data movement during GC. TTflash may alleviate write-induced read performance degradation. Unfortunately, TTflash falls short on curtailing the number of GCs, and TTflash performs poorly on write-intensive workloads.
- 2) *Amphibian*: Amphibian strives to minimize request latency by adjusting request queues. The design behind Amphibian is to avert the access conflict between I/Os and GC requests by selectively servicing I/O requests: Amphibian delays issuing requests to GC-ing locations. However, Amphibian fails to relieve GC occurrences.

TTflash and Amphibian are novel schemes proposed to alleviate the latency problem caused by GC. In particular, TTflash utilizes RAID structure and IDM-based GC to reduce GC-blocked requests, whereas Amphibian adjusts the order of I/O requests to avoid accessing the GC-ing plane. Although the two schemes utilize IDM-based GC to achieve better performance, they fail to pay attention to the storage utilization degradation caused by the parity restriction of IDM-based GC. Fortunately, our PcGC adjusts the migration order of valid pages during GC to reduce wasted pages, thereby improving

TABLE II
I/O CHARACTERISTICS OF NINE WORKLOADS

Trace	Total I/Os	Write Ratio	AWS (LSN)	WS (G)	RS (G)
<i>Mds_0</i>	1211034	0.881	14.47	7.36	3.25
<i>Hm_0</i>	3993313	0.645	16.62	20.41	9.93
<i>Ust_0</i>	2237888	0.596	20.57	13.08	35.28
<i>Web_0</i>	2029944	0.701	17.18	11.65	17.34
<i>Web_1</i>	160888	0.459	18.43	0.65	3.81
<i>Prn_0</i>	5585881	0.892	19.02	45.19	13.12
<i>Stg_0</i>	2030914	0.848	18.38	15.09	7.33
<i>Src2_0</i>	1557814	0.887	14.18	9.34	1.36
<i>Financial_1</i>	5334949	0.768	7.45	14.55	2.65

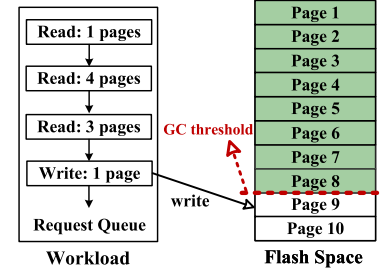


Fig. 10. Actual WS on a workload.

space utilization. Since PcGC only affects the migration order during GC, it does not conflict with the optimization of TTflash and Amphibian. This salient feature enables us to implement PcGC in TTflash and Amphibian to verify the compatibility of PcGC.

Workloads: We adopt nine realistic workloads to scrutinize the performance and durability of PcGC. Table II lists the characteristics of these nine workloads, each of which is characterized by total I/O requests, write ratio, average WS (AWS), WS, and read size (RS). These workloads contain various workload types, among which *Financial_1*—with an AWS of 4k—is collected from an online transaction processing (OLTP) application running at a financial institution [36]. The other eight workloads are obtained from Microsoft servers [37]. In addition, *Web_1* is a read-dominant workload, whereas the others are write-dominant workloads.

Among these workloads, the WS of some workloads is less than the GC trigger threshold of the simulated SSD. However, it is noteworthy that the WS of these workloads cannot represent the actual WS of flash memory, and the actual WS is the most dominant factor affecting the GC triggering. Specifically, since our workloads are the realistic workloads collected over a certain period of time, some data in flash memory is not written by the workloads’ write requests. In particular, some data may have been written into the flash chip even before the workload is executed, and then accessed by the workload’s read requests. Taking Fig. 10 as an example, the first three requests on the workload are all read requests, and the data of the read requests has been written to pages 1–8 in the flash space before executing the workload. Then, after the fourth request writes a page to the flash, the remaining space of the flash will exceed the GC threshold. Thus, the WS of the workload is one page, while the actual WS of the flash is nine pages. In other words, the actual WS of the simulated SSD is more than the WS of workloads, and both WS and RS affect the actual WS.

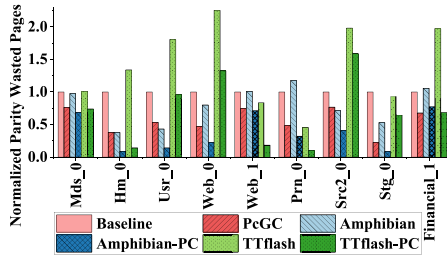


Fig. 11. Wasted pages of different schemes with nine traces.

Furthermore, the actual WS of some workloads is inadequate to generate sufficient GC operations. Fortunately, SSDsim provides the *make_aged* function that simulates old SSDs by reducing free space, which solves the problem of insufficient GC operations. Since old SSDs usually contain a large number of invalid pages that have not been collected by GC operations, we employ the *make_aged* function to add some invalid pages to the simulated SSD, thereby simulating an old SSD. The increase of invalid pages further reduces the free space of SSDs. And insufficient free space can facilitate generating sufficient GC operations for workloads with lower WSs. Therefore, in our experiments, the nine workloads with different characteristics generate sufficient GC operations, and thus effectively evaluate the performance improvement of SSDs with different optimization strategies.

Metrics: We evaluate the performance and durability of SSD with five metrics: 1) wasted pages; 2) storage utilization; 3) GC counts; 4) write amplification; and 5) average response time. Among these metrics, wasted pages and storage space intuitively indicate used space and remaining space of SSDs; GC counts and write amplification are significant indicators of SSDs' lifetime and reliability; average response time reflects the latency for an SSD to respond to I/O requests.

B. Performance Evaluation

1) *Number of Wasted Pages:* Recall from Section IV that an increasing number of wasted pages exacerbate SSD performance degradation. In this group of experiments, we investigate the impact of PcGC on the number of wasted pages under the nine workload conditions. We compare our PcGC with the other five alternative solutions, including the baseline solution, Amphibian, Amphibian-PC, TTflash, and TTflash-PC.

Fig. 11 shows the number of wasted pages of the six schemes handling the nine workload traces. Fig. 11 reveals that PcGC-based schemes noticeably curb the number of wasted pages. For example, PcGC slashes the number of wasted pages of the baseline solution by 23.6%, 61.9%, 46.6%, 52.9%, 27.4%, 51.2%, 23.3%, 77.6%, and 32.4% under conditions *Mds_0*, *Hm_0*, *Usr_0*, *Web_0*, *Web_1*, *Pron_0*, *Src2_0*, *Stg_0*, and *Financial_1*, respectively. Moreover, Fig. 11 unveils PcGC's outstanding compatibility to diminish the number of wasted pages. That is, compared with the original solutions Amphibian and TTflash, PcGC-based Amphibian-PC and TTflash-PC deliver considerable improvements in terms of reducing waste pages. For example, Amphibian-PC achieves the highest performance on all the traces except *Web_1*, *Pron_0*, and *Financial_1*. These results

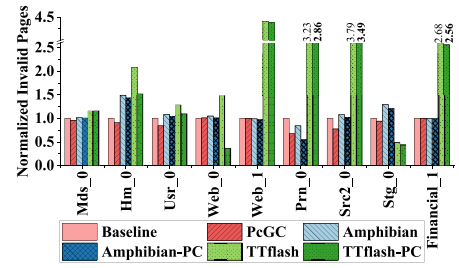


Fig. 12. Invalid pages of different schemes with nine traces.

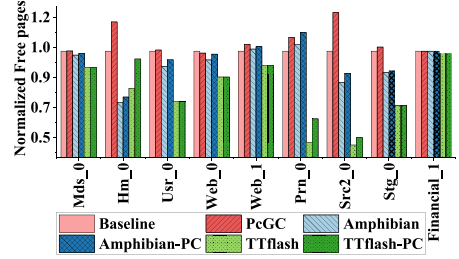


Fig. 13. Free pages of different schemes with nine traces.

demonstrate that PcGC is superior to the other alternative schemes. Unlike the existing schemes, our PcGC technique, avoiding page waste caused by the parity restriction of IDM-based GC, is adroit at boosting the utilization of the SSD space and cutting down the number of GCs.

2) *Storage Utilization:* NAND-flash SSDs have three types of pages: 1) valid pages; 2) invalid pages; and 3) free pages. Wasted pages are marked as invalid pages, whereas free pages represent unused space in SSDs. In this section, we delve into the impacts of the number of invalid and free pages on storage utilization. PcGC is competing against the other five schemes in regard to invalid and free pages under the nine workloads. Figs. 12 and 13 plot the comparison results.

Fig. 12 unravels the number of the invalid pages of the six SSD optimization schemes (few invalid pages indicate better storage utilization). We can observe from Fig. 12 that PcGC-based schemes outperform the other three counterparts in terms of invalid pages. PcGC generates the fewest invalid pages under the *Mds_0*, *Hm_0*, *Usr_0*, and *Src2_0* workload; and PcGC diminishes the invalid pages of the baseline scheme under these four traces by 4.4%, 9%, 16.6%, and 22.4%, respectively. Fig. 13 unfolds the number of the free pages of the different schemes under the same conditions—a large number of free pages indicates good storage utilization. Unsurprisingly, the PcGC-based schemes have a leading edge over the other three schemes regardless of the workload conditions. For example, PcGC gives birth to numerous of free pages under the *Hm_0* and *Src2_0* load. Figs. 12 and 13 also show that Amphibian and TTflash exhibit extremely low storage utilization, generating an excessive number of invalid pages and a limited number of free pages. Fortunately, PcGC alleviates the space waste in Amphibian and TTflash. The results confirm that PcGC is readily embedded into the other optimization schemes to restrain space waste.

3) *GC Counts:* In this set of experiments, we explore the impacts of PcGC on SSD lifetime by comparing PcGC with

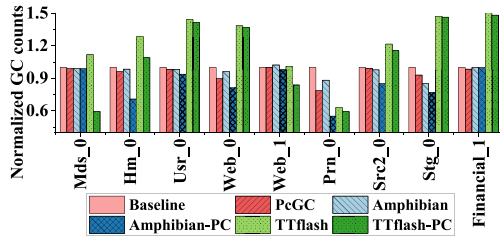


Fig. 14. GC counts of different schemes with nine traces.

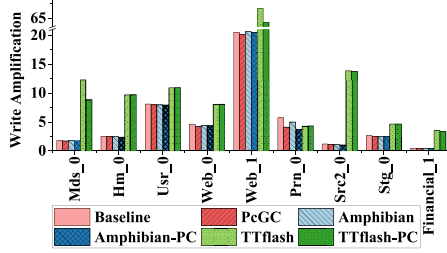


Fig. 15. Write amplification of different schemes on nine different traces.

the other five schemes in terms of GC counts. Multiple erase operations degrade the performance and reliability of NAND flash-based SSDs. Since each GC involves a block erase operation, GC counts become an effective indicator of the lifetime of SSDs—the lower the GC count, the longer the SSD lifetime.

Fig. 14 plots the GC counts corresponding to all the tested SSD optimization schemes driven by the nine real-world traces. The empirical study demonstrates that compared with the other schemes, the PcGC-based schemes are the most conducive solution to prolong the lifetime of SSDs. For instance, the PcGC-based scheme Amphibian-PC produces the least number of GC operations in the case of the five traces—*Hm_0*, *Usr_0*, *Web_0*, *Pron_0*, *Src2_0*, and *Stg_0*. Fig. 14 also suggests that TTflash has extremely high GC counts due to its stripe structure nature. Satisfyingly, with PcGC integrated into TTflash, the hybrid variant TTflash-PC is adept at slashing the number of GC operations to bolster system performance. When it comes to workload *Mds_0*, PcGC is capable of helping TTflash to cut back GC operations by 52.2%. Due to the fact that workload *Financial_1* has a small size of I/O data, the GC performance of the PcGC-based schemes are close to those of the other schemes managing *Financial_1*. In short, these results are expected: PcGC improves the storage space utilization by trimming the number of wasted pages. Such an improvement directly lessens the number of GCs, thereby prolonging SSD lifetime.

4) *Write Amplification*: In this section, we evaluate the write amplification of the six SSD optimization schemes driven by the nine different traces. Write amplification represents the ratio between the number of actual write accesses and the number of write I/Os issued by a host. Write amplification serves as an important and viable indicator of SSD overhead—small write amplification signifies low SSD overhead, and the reason is that the amount of actual written data is susceptible to data migration and the 4k misalignment constrain.

Fig. 15, plotting the write amplification of different schemes, reveals that PcGC claims a winning position among

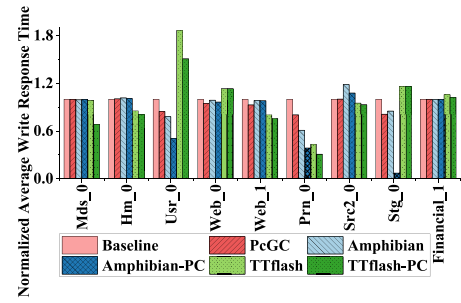


Fig. 16. Average write response time with different schemes.

its peers with respect to write amplification under workloads *Web_0*, *Web_1*, *Pron_0*, and *Stg_0*. Fig. 15 also demonstrates that TTflash-PC considerably reduces the write amplification of the original TTflash under workloads *Mds_0* and *Web_1*. When running on the other traces, PcGC marginally suppresses the write amplification of the other schemes. Among these six schemes, TTflash delivers the worst write amplification performance because TTflash is designed on the basis of RAID parity, which induces extra write requests.

These experimental results suggest that PcGC, offering the lowest write amplification, is a clear champion in the competition against the six alternative schemes. PcGC enjoys its superiority in low write amplification because PcGC is adept at bolstering space utilization and bringing down GC counts, which alleviates the write amplification problem.

5) *Average Response Time*: In the last set of experiments, we are in a position to evaluate the average response time of PcGC. Again, we compare PcGC with the other five alternative schemes governing 3-D NAND flash-based SSDs running the nine traces.

Fig. 16, illustrating the average write response time of the six schemes on the nine traces, unveils that PcGC-based schemes have a performance edge over the other schemes driven by the nine workloads. For instance, PcGC slashes the write response time of the baseline scheme by 15.4%, 6.1%, 19.1%, and 18.9% in the case of *Usr_0*, *Web_0*, *Pron_0*, and *Stg_0*, respectively. The PcGC-based scheme TTflash-PC exhibits the best performance on traces *Mds_0*, *Hm_0*, *Web_1*, *Pron_0*, and *Src2_0*. Moreover, TTflash has inadequate write latency performance in some traces because stripe-based TTflash generates extra read/write requests when processing write requests. Meanwhile, the PcGC-based scheme Amphibian-PC achieves the best performance on traces *Usr_0* and *Stg_0*. It can be seen that the space saved by PcGC can significantly improve flash performance. Under workload *Financial_1*, PcGC's performance is on par with the other schemes because of several reasons. First, a small number of GCs does not necessarily result in high I/O latency, whereas a large number of requests in a short time will definitely lead to extremely high latency. Second, the AWS of *Financial_1* is fixed at 4k, and the corresponding 4K-alignment constraint noticeably enlarges I/O latency.

Fig. 17 shows the average read response time of six schemes under nine workloads, which reveals that the PcGC-based schemes outperform the other schemes in terms of read response time. For example, PcGC reduces the average read

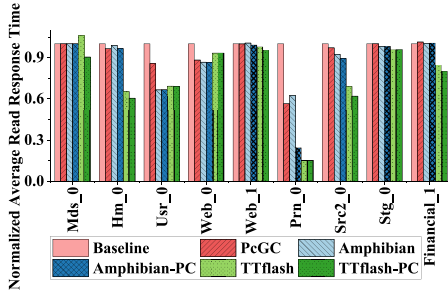


Fig. 17. Average read response time with different schemes.

response time of the baseline scheme by 4.6%, 14.5%, 11.9%, 43.6%, and 2.9% on traces *Hm_0*, *Usr_0*, *Web_0*, *Prn_0*, and *Src2_0*, respectively. Additionally, TTflash has a decent performance in handling read requests due to the parity RAID structure. However, the PcGC-based hybrid scheme, TTflash-PC, has the most significant performance on seven traces. Specifically, compared with the original TTflash, TTflash-PC lowers the average read response time by 10.2%, 5.1%, 8.9%, and 4.5% on traces *Mds_0*, *Hm_0*, *Src2_0*, and *Financial_1*, respectively. Likewise, Amphibian-PC also has lower response time than the original scheme. In particular, under trace *Prn_0*, Amphibian-PC reduces the average read response time by 38% compared with Amphibian. These results are expected: GC is a root cause of extended read/write latency; PcGC can effectively alleviate page waste to increase the storage space utilization of SSDs, thereby slashing GCs. Therefore, the PcGC-based schemes can alleviate GC-blocked I/Os and further reduce read and write latency.

In a nutshell, the above experiments confirm that compared with the existing Amphibian and TTflash schemes, PcGC is conducive to boosting both the performance and endurance of NAND flash-based SSDs. For example, PcGC brings down the number of wasted pages of Amphibian and TTflash by 40.2% and 67.3%, respectively. PcGC also cuts back the number of GCs of Amphibian and TTflash by 11.6% and 12.2%, respectively. Moreover, PcGC exhibits prominent performance in regards to storage space utilization and average response time. Furthermore, the experimental results of PcGC-based hybrid solutions Amphibian-PC and TTflash-PC unfold that our PcGC can be integrated into the existing solutions to boost the performance of NAND flash-based SSDs.

VII. CONCLUSION

In this article, we proposed a parity-check GC scheme, coined as PcGC, to optimize the space utilization and performance of solid-state disks. We integrated a parity-check unit into PcGC to facilitate checking the parity of migrated valid pages and destination pages. According to the parity results generated by the parity-check unit, PcGC dynamically adjusts the migration order of valid pages, thereby averting page waste during the course of GC. As a result, PcGC significantly expands free space in flash chips and restrains the number of GCs.

We conducted the extensive experiments to quantitatively gauge the performance of PcGC by comparing our solution

with a conventional GC scheme and two state-of-the-art techniques—TTflash and Amphibian. The experimental results unveil that PcGC curtails the number of wasted pages and average write response time by an average of 53.75% and 13.0%, respectively. Furthermore, our findings confirm that PcGC is conducive to a seamless integration with the other GC-optimization schemes to revamp the performance of 3-D NAND flash-based SSDs.

REFERENCES

- [1] N. Shahidi, M. T. Kandemir, M. Arjomand, C. R. Das, M. Jung, and A. Sivasubramanian, "Exploring the potentials of parallel garbage collection in SSDs for enterprise storage systems," in *Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal.*, 2016, pp. 561–572.
- [2] C.-Y. Liu, J. B. Kotra, M. Jung, and M. T. Kandemir, "PEN: Design and evaluation of partial-erase for 3D NAND-based high density SSDs," in *Proc. 16th USENIX Conf. File Storage Technol.*, 2018, pp. 67–82.
- [3] G. Zhang, Y. Deng, Y. Zhou, S. Pang, J. Yue, and Y. Zhu, "Cocktail: Mixing data with different characteristics to reduce read reclaims for NAND flash memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Oct. 14, 2022, doi: [10.1109/TCAD.2022.3214679](https://doi.org/10.1109/TCAD.2022.3214679).
- [4] "NAND flash performance improvement using internal data move." Micron. 2006. [Online]. Available: <http://download.micron.com/pdf/technotes/tn2915.pdf>
- [5] 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. 25th Int. Conf. Supercomput.*, 2011, pp. 96–107.
- [6] *NAND Memory Toggle DDR1.0 Technical Data Sheet*, TOSHIBA, Tokyo, Japan, 2013.
- [7] "Open NAND flash interface specification revision 5.0." ONFI. 2021. [Online]. Available: <https://www.onfi.org/specifications>
- [8] *3D Flash Memory Toggle DDR2.0 Technical Data Sheet*, TOSHIBA, Tokyo, Japan, 2016.
- [9] "1Gb 3.3V x1, x2, x4: SPI NAND flash memory data sheet." Micron. 2022. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/nand-flash/70-series/m78a_1gb_3v_nand_spi.pdf?rev=2fc0af24f358402ca678ba5a0212a18e
- [10] "4Gb 3.3V x1, x2, x4: SPI NAND flash memory data sheet." Micron. 2016. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/nand-flash/70-series/m70a_4gb_3v_nand_spi.pdf?rev=ca0cf6d7dc0848c3808fbc1ca1b740d9
- [11] "2Gb 1.8V, x1, x2, x4: SPI NAND flash memory data sheet." Micron. 2022. [Online]. Available: https://www.micron.com/-/media/client/global/documents/products/data-sheet/nand-flash/70-series/m79a_2gb_1_8v_nand_spi.pdf?rev=a4c59654640d40b3a12365c2ee09c81d
- [12] W. Kang, D. Shin, and S. Yoo, "Reinforcement learning-assisted garbage collection to mitigate long-tail latency in SSD," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–20, 2017.
- [13] L.-P. Chang, T.-W. Kuo, and S.-W. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Trans. Embedded Comput. Syst.*, vol. 3, no. 4, pp. 837–863, 2004.
- [14] G. Wu and X. He, "Reducing SSD read latency via NAND flash program and erase suspension," in *Proc. 10th USENIX Conf. File Storage Technol.*, 2012, p. 10.
- [15] J. Lee, Y. Kim, G. M. Shipman, S. Oral, F. Wang, and J. Kim, "A semi-preemptive garbage collector for solid state drives," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw.*, 2011, pp. 12–21.
- [16] Z. Sha et al., "Low I/O intensity-aware partial GC scheduling to reduce long-tail latency in SSDs," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, pp. 1–25, 2021.
- [17] B. Mao, S. Wu, and L. Duan, "Improving the SSD performance by exploiting request characteristics and internal parallelism," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 2, pp. 472–484, Feb. 2018.
- [18] S. Yan et al., "Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs," in *Proc. 15th USENIX Conf. File Storage Technol.*, 2017, pp. 15–28.

- [19] W. Wang and T. Xie, "PCFTL: A plane-centric flash translation layer utilizing copy-back operations," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3420–3432, Dec. 2015.
- [20] D. Hong, M. Kim, J. Park, M. Jung, and J. Kim, "Improving SSD performance using adaptive restricted-copyback operations," in *Proc. IEEE Non-Volatile Memory Syst. Appl. Symp.*, 2019, pp. 1–6.
- [21] W. Chang, Y. Lim, and J. Cho, "An efficient copy-back operation scheme using dedicated flash memory controller in solid-state disks," *Int. J. Electr. Energy*, vol. 2, no. 1, pp. 13–17, 2014.
- [22] G. Naso et al., "A 128Gb 3b/cell NAND flash design using 20nm planar-cell technology," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, 2013, pp. 218–219.
- [23] C.-H. Hung et al., "Layer-aware program-and-read schemes for 3D stackable vertical-gate BE-SONOS NAND flash against cross-layer process variations," *IEEE J. Solid-State Circuits*, vol. 50, no. 6, pp. 1491–1501, Jun. 2015.
- [24] M. Zhang, F. Wu, Y. Du, W. Liu, and C. Xie, "Pair-bit errors aware LDPC decoding in MLC NAND flash memory," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 12, pp. 2312–2320, Dec. 2019.
- [25] W.-C. Chen, H.-T. Lue, C.-C. Hsieh, K.-C. Wang, and C.-Y. Lu, "Performance enhancement of 3-D NAND flash featuring a two-step dummy wordline program waveform and pair-bitline program scheme," *IEEE Trans. Electron Devices*, vol. 67, no. 1, pp. 99–104, Jan. 2020.
- [26] Z. Shi, Y. Fang, Y. Bu, and P. Chen, "Mitigation of cell-to-cell interference in multi-level-cell (MLC) NAND flash memory using correction factors," in *Proc. IEEE/CIC Int. Conf. Commun. China*, 2021, pp. 629–634.
- [27] J. Guo, W. Wen, J. Hu, D. Wang, H. Li, and Y. Chen, "FlexLevel: A novel NAND flash storage system design for LDPC latency reduction," in *Proc. 52nd ACM/IEEE Des. Autom. Conf.*, 2015, pp. 1–6.
- [28] J. Jang et al., "Vertical cell array using TCAT(Terabit cell array transistor) technology for ultra high density NAND flash memory," in *Proc. IEEE Symp. VLSI Technol.*, 2009, pp. 192–193.
- [29] W. Jeong et al., "A 128 Gb 3b/cell V-NAND flash memory with 1 Gb/s I/O rate," *IEEE J. Solid-State Circuits*, vol. 51, no. 1, pp. 204–212, Jan. 2016.
- [30] M.-C. Yang, Y.-M. Chang, C.-W. Tsao, P.-C. Huang, Y.-H. Chang, and T.-W. Kuo, "Garbage collection and wear leveling for flash memory: Past and future," in *Proc. Int. Conf. Smart Comput.*, 2014, pp. 66–73.
- [31] C. Kim et al., "A 512-Gb 3-b/cell 64-stacked WL 3-D-NAND flash memory," *IEEE J. Solid-State Circuits*, vol. 53, no. 1, pp. 124–133, Jan. 2018.
- [32] S. Wu, C. Du, H. Li, H. Jiang, Z. Shen, and B. Mao, "CAGC: A content-aware garbage collection scheme for ultra-low latency flash-based SSDs," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2021, pp. 162–171.
- [33] W. Kang and S. Yoo, "Q-value prediction for reinforcement learning assisted garbage collection to reduce long tail latency in SSD," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2240–2253, Oct. 2020.
- [34] Q. Xiong et al., "Characterizing 3D floating gate NAND flash: Observations, analyses, and implications," *ACM Trans. Storage*, vol. 14, no. 2, pp. 1–31, 2018.
- [35] W. Zhang, Q. Cao, H. Jiang, and J. Yao, "Improving overall performance of TLC SSD by exploiting dissimilarity of flash pages," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 332–346, Feb. 2020.
- [36] "UMASS trace repository." Laboratory for Advanced System Software. 2007. [Online]. Available: <http://traces.cs.umass.edu/index.php/Storage/Storage>
- [37] "MSR Cambridge traces." SNIA IOTTA. 2007. [Online]. Available: <http://iota.snia.org/traces/block-io/388?only=386>



Shujie Pang received the M.S. degree from the Computer Science Department, Yantai University, Yantai, China, in 2020. He is currently pursuing the Ph.D. degree with Computer Science Department, Jinan University, Guangzhou, China.

His current research interests include storage system, computer architecture, and performance evaluation.



Yuhui Deng received the Ph.D. degree in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 2004.

He is a Professor with the Department of Computer Science, Jinan University, Guangzhou, China. Before joining Jinan University, he worked with EMC Corporation, Hopkinton, MA, USA, as a Senior Research Scientist from 2008 to 2009. He worked as a Research Officer with Cranfield University, Cranfield, U.K., from 2005 to 2008. His research interests cover information storage, computing, computer architecture, and performance evaluation.



Genxiong Zhang received the B.E. degree in computer science and technology from South China Agricultural University, Guangzhou, China, in 2020.

He is a Research Student with the Computer Science Department, Jinan University, Guangzhou. His current research interests include storage system and nonvolatile memory.



Yi Zhou received the B.E.E. and M.S.E.E. degrees in electronic engineering from the Beijing University of Technology, Beijing, China, in 2006 and 2010, respectively, and the Ph.D. degree in computer science from Auburn University, Auburn, AL, USA, in 2018.

He is currently an Assistant Professor with the TSYS School of Computer Science, Columbus State University, Columbus, GA, USA. His research interests include energy-saving techniques, database systems, big data techniques, and parallel computing.



Xiao Qin received the B.S. and M.S. degrees in computer science from the Huazhong University of Science and Technology, Wuhan, China, in 1992 and 1999, respectively, and the Ph.D. degree in computer science from the University of Nebraska–Lincoln, Lincoln, NE, USA, in 2004.

He is currently an Alumni Professor and the Director of Graduate Programs with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA. His research interests include parallel and distributed systems, storage systems, and performance evaluation.

Dr. Qin won an NSF CAREER Award in 2009.



Zhaorui Wu received the M.S. degree in computer architecture from the Computer Science Department, Jinan University, Guangzhou, China, in 2021, where he is currently pursuing the Ph.D. degree.

His current research interests include parallel and distributed computing, computer architecture, and performance evaluation.



Jie Li received the B.E. degree in automation from the Wanjia University of Technology, Maanshan, Anhui, China, in 2016, and the M.S. degree in computer science and technology from the Guangxi University for Nationalities, Nanning, China, in 2019. He is currently pursuing the Ph.D. degree with the Computer Science Department, Jinan University, Guangzhou, China.

His current research interests include parallel and distributed computing, data center architecture, cloud computing, and data replica placement.