SCORE: A Novel Scheme to Efficiently Cache Overlong ECCs in NAND Flash Memory

YOU ZHOU and FEI WU, Huazhong University of Science and Technology, China

ZHONGHAI LU, KTH Royal Institute of Technology, Sweden

XUBIN HE and PING HUANG, Temple University, USA

CHANGSHENG XIE, Huazhong University of Science and Technology, China

Technology scaling and program/erase cycling result in an increasing bit error rate in NAND flash storage. Some *solid state drives (SSDs)* adopt *overlong error correction codes (ECCs)*, whose redundancy size exceeds the spare area limit of flash pages, to protect user data for improved reliability and lifetime. However, the read performance is significantly degraded, because a logical data page and its ECC redundancy are stored in two flash pages. In this article, we find that caching ECCs has a large potential to reduce flash reads by achieving higher hit rates, compared to caching data. Then, we propose a novel scheme to efficiently cache overlong ECCs, called *SCORE*, to improve the SSD performance. Exceeding ECC redundancy (called *ECC residues*) of logically consecutive data pages are grouped into *ECC pages*. SCORE partitions RAM to cache both data pages and ECC pages in a workload-adaptive manner. Finally, we verify SCORE using extensive trace-driven simulations. The results show that SCORE obtains high ECC hit rates without sacrificing data hit rates, thus improving the read performance by an average of 22% under various workloads, compared to the state-of-the-art schemes.

CCS Concepts: • **Information systems** → **Flash memory**; • **Hardware** → *External storage; Error detection and error correction; Memory and dense storage*; • **Theory of computation** → *Caching and paging algorithms*; • **Computer systems organization** → *Firmware*; • **General and reference** → Performance;

Additional Key Words and Phrases: Solid state drive, overlong ECC, cache partitioning

New Paper, Not an Extension of a Conference Paper.

This research is sponsored by the National Natural Science Foundation of China (Grants No. 61872413, No. U1709220, and No. 61821003), Wuhan Science and Technology Project (Grant No. 2017010201010108), the Fundamental Research Funds for the Central Universities (Grant No. 2016YXMS019), the 111 Project (Grant No. B07038), the Shenzhen Basic Research Project (Grants No. JCYJ20170307160135308 and No. JCYJ20170818162129916), and the Key Laboratory of Data Storage System, Ministry of Education. The work performed at Temple is partially sponsored by the U.S. National Science Foundation under Grants No. 1702474 and No. 1717660.

Authors' addresses: Y. Zhou, School of Optical and Electronic Information and Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China; email: zhouyou@hust.edu.cn; F. Wu (corresponding author), Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China; email: wufei@hust.edu.cn; Z. Lu, Department of Electronic Systems, KTH Royal Institute of Technology, Stockholm, Sweden; email: zhonghai@kth.se; X. He, and P. Huang, Department of Computer and Information Sciences, Temple University, Philadelphia, PA, 19122, USA; emails: {xubin.he, templestorager}@temple.edu; C. Xie, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, China; email: cs_xie@hust.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

1544-3566/2018/12-ART60

<https://doi.org/10.1145/3291052>

ACM Reference format:

You Zhou, Fei Wu, Zhonghai Lu, Xubin He, Ping Huang, and Changsheng Xie. 2018. SCORE: A Novel Scheme to Efficiently Cache Overlong ECCs in NAND Flash Memory. *ACM Trans. Archit. Code Optim.* 15, 4, Article 60 (December 2018), 25 pages.
<https://doi.org/10.1145/3291052>

1 INTRODUCTION

NAND flash memory has become prevalent in various systems as persistent storage medium, such as embedded systems, smartphones, personal computers, and data centers. This trend is driven by aggressive flash technologies scaling over the past 20 years, such as more bits stored per cell (from one bit to two/three/four bits) [53], smaller feature sizes (from larger than 100nm to 1Znm) [59], and 3D structures with more stacked layers (from 24 layers in 2013 to 96 layers in 2018) [25, 36, 67]. Such scaling provides increasing storage density and decreasing bit cost, which are expected to continue in the future.

However, flash memory are vulnerable to bit errors, caused by various types of noises, such as *program/erase (P/E)* cycling, charge leakage (i.e., the data retention issue), and read/program disturbances [8, 10, 12]. On the one hand, flash technologies scaling results in degraded reliability [22, 61, 76]. On the other hand, the *raw bit error rate (RBER)* of flash memory grows dramatically as the number of P/E cycles or the retention time increases. Specifically, the RBER can grow by up to 2000 times and beyond 10^{-2} as P/E cycles are consumed [21, 41]. A recent study shows that the RBER of a state-of-the-art 3D flash memory is larger than 10^{-2} for one year's retention under 1,500 P/E cycles¹ [73]. Therefore, high-density flash memory suffers from a serious reliability problem.

To improve the reliability of flash storage, *error correction codes (ECCs)* are used to detect and correct bit errors by adding parity redundancy [79]. The ratio of data size to the total size of data and its ECC redundancy is defined as ECC code rate. A lower ECC code rate indicates stronger error correction capability. A qualifying ECC should guarantee the *uncorrectable bit error rate (UBER)* of flash storage is under an acceptable level, such as 10^{-15} or 10^{-17} . Note that flash memory performs read/write operations in a unit of pages. Each flash page contains a *data area* and a *spare area* and their sizes are predefined by the flash chip manufacturer. In the latest 3D and planar flash chips, the ratio of spare area size to data area size is about 1/8, such as 1,024 bytes to 8KB and 1,952 bytes to 16KB [4, 50, 64]. The ECC redundancy of the page data, called an *ECC unit*, is stored in the spare area (as shown in Figure 1), which is transparent to users. The spare area size could be a limit to the effective ECC code rate and thus the error correction capability, while simply enlarging the spare area is not a practically viable approach as it increases the manufacture cost.

Researches from both industry [6, 21, 26] and academia [17, 28, 29, 35, 38, 66] reveal that some flash-based *solid state drives (SSDs)* employ low code rate ECCs, whose ECC unit size exceeds the spare area limit, called *overlong ECCs*. We name the exceeding part of an ECC unit as an *ECC residue*, as shown in Figure 1. BCH codes and low-density parity-check (LDPC) codes are two commonly used ECC algorithms in modern SSDs [79]. Generally, effective LDPC code rates range from 0.75 to 0.95 [30, 77], while BCH code rates are lower, because BCH codes offer weaker error correction capability than LDPC codes. Considering a typical spare area size, a code rate smaller than 0.89 (e.g., $8/(8+1)$) leads to an overlong ECC. We justify why overlong ECCs are used as follows.

First, users may have high and complicated reliability requirements for SSDs. An overlong ECC can largely reduce the UBER to achieve higher storage reliability [26, 38] and prolong the storage

¹The JEDEC standard requires at least one year's data retention at 30°C under the maximum P/E cycles specified for client SSDs [3].

Fig. 1. **Data page layouts with different ECCs.** An *ECC unit* refers to the ECC redundancy of a logical data page. A *regular* ECC unit can be held in the spare area along with the protected data in a flash page, while an *overlong* ECC unit is beyond the spare area size. We call the exceeding part of an ECC unit an *ECC residue*.

Fig. 2. **Data Layout of Cross-Page and PG Decoupling.** Suppose a flash page includes 8KB data area and 1KB spare area; a logical page has 8KB data and 1.5KB ECC unit (the code rate is 0.84). In Cross-Page, each logical page and its ECC unit are stored in two flash pages. In PG Decoupling, ECC residues of 18 physically consecutive data pages are grouped into one ECC page. ECC pages are stored in a dedicated flash area.

lifetime (i.e., endure higher P/E cycles) [17, 27, 66] and data retention time. Second, the RBER of flash memory largely increases during the lifetime as P/E cycles are consumed. Reserving a large enough spare area for the worst case ECC would greatly increase the flash manufacture cost. Choosing a proper ECC code rate is a tradeoff between the error correction capability and overhead, such as redundancy storage and decoding latency. Thus, rate-adaptive ECC schemes are proposed to better balance the performance and lifetime of SSDs [6, 21, 29, 66]. The ECC code rate is gradually lowered over time, resulting in an overlong ECC at the late stage. Third, inefficient ECC algorithms may be used due to cost and intellectual property limitations. Assuming the RBER of flash memory is 10^{-2} and the ECC engine performs encoding/decoding in a unit of 2KB, the BCH code rate that ensures a UBER of no larger than 10^{-15} can be calculated as 0.78 [18].

Adopting an overlong ECC for user data is an effective solution to the reliability concern of high-density SSDs. However, how to manage ECC units becomes a problem, since a logical data page and its overlong ECC unit cannot be stored in one flash page. Existing solutions can be classified into two categories. *Coupling schemes* store a data page and its ECC unit as a whole. For example, Jung et al. proposed to shrink the data page size by storing less data in each flash page [35]; Wang et al. proposed to store a data page and its ECC unit in a cross-page fashion, called *Cross-Page* [66] (shown in Figure 2(a)). *Decoupling schemes* separately store data pages and their ECC residues. For example, ECC residues are grouped into *ECC pages* [17, 29] or are stored in an extra non-volatile storage chip [38]. We name the decoupling scheme, where ECC residues of multiple physically

consecutive data pages are grouped into ECC pages and stored in a reserved flash area, as *PG Decoupling* (shown in Figure 2(b)).

Among these schemes, Cross-Page and PG Decoupling are practical for general SSDs. However, they suffer from a significant read performance degradation, because a logical data page and its ECC unit are stored in two flash pages. In this article, we first find that caching ECC residues has a large potential to reduce flash reads by achieving higher hit rates, compared to caching data. Then, we propose a novel scheme to efficiently cache overlong ECCs, called *SCORE*, to improve the SSD performance. ECC residues of logically consecutive data pages are grouped into *ECC pages*, which are uniformly managed with data pages. Further, SCORE partitions RAM to cache both data pages and ECC pages in a workload-adaptive manner. Finally, we verify SCORE using extensive trace-driven simulations. The results show that SCORE obtains high ECC hit rates without sacrificing data hit rates, thus significantly improving SSD read performance with a small write overhead under various workloads, compared to the state-of-the-art schemes.

The rest of the article is organized as follows. Section 2 provides an overview of background and related work. Section 3 presents our motivation and analysis. Section 4 details our design. We evaluate our design comprehensively in Section 5 and conclude the article in Section 6.

2 BACKGROUND AND RELATED WORK

2.1 NAND Flash Memory Basics

The storage rationale of NAND flash memory is that a flash cell uses the threshold voltage, determined by the number of electrons stored in it, to represent bit states. Flash cells are organized into *blocks*, each of which contains a fixed number of *pages* (e.g., 256 to 1,152). Read operations, which sense the cell's threshold voltage, and write operations, which inject electrons into the cell, are performed in a unit of a page. Erase operations, which eject the electrons stored in cells, are performed in a unit of a block. Flash memory cannot be updated in place, because a block cannot be written before it is erased. Moreover, flash memory has limited endurance/lifetime. For example, contemporary flash memory with multiple bits stored in each cell can only sustain a few hundred to a few thousand P/E cycles.

Due to such unique features, flash memory is usually managed by a *flash translation layer (FTL)* [20]. The key functions of the FTL include address mapping, garbage collection, and wear leveling [20]. The FTL performs out-of-place updates, where new data are written to free flash pages while old data become invalid. Thus, *address mapping* is needed to maintain a table that translates a *logical page number (LPN)* to a *physical page number (PPN)*. The mapping table is packed into *translation pages* and stored in flash memory for persistence [23]. To support fast address translation, the mapping table is also cached in RAM, called *mapping cache* [80]. When free flash blocks run out, *garbage collection (GC)* reclaims victim blocks containing invalid data by first migrating valid data and then erasing the blocks [74]. GC operations degrade both the performance and lifetime of an SSD. To improve the GC efficiency and compensate bad flash blocks, SSDs provide more storage space, typically 7% to 28%, than the user-visible capacity, called *over-provisioning space* [31]. *Wear leveling* distributes P/E cycles over blocks to maximize the SSD lifetime [16].

2.2 Flash Reliability Issues

Flash memory suffers from several reliability issues, such as raw bit errors and erase errors. Extensive studies have characterized these errors and some techniques are proposed to mitigate each type of errors [8–15, 44–47, 68, 73].

Cai et al. [10] characterized the error patterns of planar flash memory. Four types of errors are commonly observed: erase errors, retention errors, program interference errors, and read disturb

errors. Erase errors are due to extensive P/E cycling and fabrication defects. P/E cycling gradually accumulates defects in flash cells (i.e., trapped electrons in the tunnel oxide) and thus permanently degrades the cell reliability. A statistical model between the threshold voltage distribution and P/E cycles is developed in Reference [45]. The other three types of errors appear to be raw bit errors, which are caused by threshold voltage shifts. Retention errors, caused by electron leakage in flash cells, are characterized in Reference [12]. They can be reduced by adjusting the read reference voltage [12] and periodically refreshing the data [14, 44]. The program interference and read disturbance refer to the effects that threshold voltages of neighboring cells unintentionally change when a flash cell is programmed or read, respectively. These effects intrinsically result from the flash architecture. The program interference, caused by the parasitic capacitance coupling, is characterized and modeled in Reference [13]. By considering the program interference effect, a neighbor-cell assisted error correction technique was proposed in Reference [15]. To reduce the program interference, the multi-step programming method is adopted in planar flash memory. However, this method leaves partially-programmed cells more vulnerable [9]. The read disturbance, caused by weak programs, is characterized in Reference [11], which also proposed to reduce read disturb errors by tuning the pass-through voltage. To provide a comprehensive overview, Cai et al. [8] summarized recent advances in the error characterization, mitigation, and data recovery techniques of flash memory.

Recently, flash memory manufacturers have turned to 3D architecture to continue increasing the storage density. Different from planar flash, 3D flash stacks memory cells vertically and most 3D flash memories use charge trap transistors instead of floating gate transistors. Hence, 3D flash memory shows different characteristics, which have been demonstrated in References [68, 73]. Compared to planar flash memory, 3D charge trap flash memory is likely to mitigate the program interference and read disturbance but be more vulnerable to retention errors. Furthermore, 3D flash memory suffers from unique reliability issues, such as cross-layer error variations, early retention loss, and retention interference [47]. Luo et al. [46] characterized the effects of self-recovery and temperature on 3D flash memory and proposed HeatWatch to improve its reliability by optimizing the read reference voltage.

In addition, integrated circuit failures, which are not specific to flash memory, occasionally occur. For example, fabrication defects in the peripheral circuitry could cause die failures [51]. To prevent data loss from such failures, SSDs usually employ the die-level RAIN (redundant array of independent NAND) technique [74].

It is important to note that flash technologies are aggressively scaling to increase the storage density. First, manufacturers have been shrinking the feature size, which reaches 1Znm for planar flash memory. Second, an increasing number of layers (e.g., currently 64 or 96 layers) are vertically stacked in 3D flash memory. Third, more bits are stored in each flash cell, from *single-level cell* (SLC) to *multi-level cell* (MLC), to *triple-level cell* (TLC), and to *quad-level cell* (QLC). Due to the circuit-level and structural challenges, these technologies have been also degrading the reliability of flash cells. As a result, the reliability issues are a critical concern for high-density flash storage.

2.3 Error Correction Codes for Flash

Each of above techniques targets a specific type of bit errors. A more common and widely employed technique is the *error correction code* (ECC). ECC algorithms are able to correct multiple bit errors and lower the uncorrectable bit error rate [79]. In this article, we do *not* impose any restrictions on the ECC algorithm. Our proposed design is applicable to either BCH codes or LDPC codes as long as the ECC unit size exceeds the spare area limit. The ECC algorithm is typically implemented by hardware. To reduce hardware cost and decoding latency, the ECC engine divides data into segments, typically 2KB. A data segment and its ECC redundancy together form a

codeword. Encoding and decoding are performed in a unit of codewords and pipelined with their transferring [71]. When the FTL writes a logical page, the ECC engine encodes its data into multiple codewords. Then, the page data and its ECC unit (i.e., all the codewords) are programmed into a flash page. When the FTL reads a logical page, codewords in the relevant flash page are read to the ECC engine. The raw data can be obtained by decoding the codewords, where bit errors, if within the error correction capability, are detected and corrected. If uncorrectable bit errors occur and no other data protection policies are available, then the read is marked as a failure.

2.4 Overlong ECCs in SSDs

Overlong ECCs have been adopted in SSDs to improve the reliability and lifetime. Both coupling schemes and decoupling schemes were proposed to manage overlong ECCs.

Coupling schemes: Jung et al. proposed to include a logical data page and its ECC unit in one flash page by storing fewer codewords [35]. This scheme has two drawbacks. First, data page size is shrunk and storage space utilization decreases. This degradation is highly sensitive to flash page and ECC configurations. For example, when a flash page has 8KB data area and 1KB spare area and each codeword contains 1 or 2KB data, data page size is reduced from 8 to 7KB (i.e., seven codewords) or 6KB (i.e., three codewords), respectively. Since a smaller codeword size with the same code rate result in weaker error correction capability [71], reducing the codeword size is not effective to alleviate this problem. Second, unaligned 4KB data page size probably incurs cross-page accesses. Hence, shrinking the data area has poor applicability and performance.

Wang et al. proposed to adopt a rate-adaptive ECC scheme during the lifetime of an SSD [66]. As the number of P/E cycles increases, the ECC code rate gradually becomes lower to ensure reliability and improve lifetime at a small performance overhead. Finally, an overlong ECC is employed, where data pages and their ECC units are stored using the Cross-Page scheme. Figure 2(a) shows the data layout in Cross-Page. A logical page and its ECC unit are stored in two flash pages and a flash page stores data from at most two logical pages (this limitation avoids a logical page being stored in multiple flash pages, which would significantly complicate the design and degrade the read performance).

Decoupling schemes: Chang et al. proposed to employ two levels of ECCs [17]. If a data page read fails using a weak ECC, whose redundancy is stored in the spare area, then a strong ECC is invoked to recover the data. The strong ECC's redundancy is managed using the PG Decoupling scheme, as shown in Figure 2(b). Accessing ECC pages is not efficient in such a scheme due to the physical grouping method, as discussed in Section 4.2. Some designs were proposed to store ECC residues in dedicated high-speed non-volatile storage medium, such as SLC flash [29] and phase change memory (PCM) [38]. However, using SLC or PCM brings a high hardware cost. MLC or TLC flash can be operated in SLC mode [58], but the capacity would be reduced by a half or two thirds, respectively. Assume the logical page size is 8KB and ECC residue size is 0.5 or 1KB. If we use the SLC mode of TLC flash to store ECC residues of all user data, then the capacity loss would be 12.5% or 25% of the user-visible capacity, respectively. By contrast, PCM is five times more expensive than NAND flash [54]. If we use PCM to store ECC residues of all user data, then the extra cost would be more than 31.3% or 62.5% of the flash cost, respectively. To reduce the cost, [38] targets the hybrid mapping scheme (not the widely used page-level mapping) and uses PCM to store ECC units of only log blocks (i.e., a small number of flash blocks).

In addition, some designs use an overlong ECC to protect some specific flash pages [77] or blocks [27, 28] with high RBERs. For example, an overlong ECC and a weak ECC are used for heavily and lightly worn flash blocks, respectively, to reduce the wear leveling overhead in Reference [28]. The overlong ECC redundancy is stored in dedicated pages of target blocks.

Fig. 3. **Read performance degradation caused by the overlong ECC.** The read amplification of Cross-Page and PG Decoupling is mainly because a logical data page and its ECC unit are stored in two flash pages.

In this article, we target SSDs that adopt an overlong ECC for all user data (not just for some specific flash pages/blocks). Among existing schemes, Cross-Page [66] and PG Decoupling [17] are applicable to general SSDs, but their read performance is poor. We aim to address this problem in this article.

3 MOTIVATION AND ANALYSIS

In this section, we first demonstrate the read performance degradation caused by adopting an overlong ECC and then explore the potential of caching ECC residues.

3.1 Read Performance Degradation

Read performance is critical for SSDs for two reasons. First, application performance highly depends on the response speed of read requests, which are caused by upper level cache misses [70]. Second, due to the promising read performance, SSDs are widely employed in different storage systems to accelerate read-intensive applications [19, 40, 60, 75]. However, *SSDs adopting an overlong ECC suffer from a significant read performance degradation*. We have conducted experiments to show the read amplifications and average read response time of three schemes, Cross-Page [66], PG Decoupling [17], and Optimal. *Read amplification*, which refers to the ratio of the data volume of flash reads to the data volume of user reads, reflects the SSD read efficiency. Optimal represents an ideal case where the spare area is large enough to hold an overlong ECC unit. The performance deviation of a scheme from Optimal indicates the overhead of adopting an overlong ECC. Detailed configurations can be found in Section 5.1. As shown in Figure 3(a), Cross-Page and PG Decoupling increase the read amplification by an average of 100.9% and 102.5%, respectively, compared to Optimal. The major reason is that a logical data page and its ECC unit are stored in two flash pages. The read amplification of Optimal can be larger than 1 because of partial page reads and flash reads caused by address translation and garbage collection. Accordingly, the average read response time is increased by an average of 58.1% and 63%, respectively, as shown in Figure 3(b).

Some existing works have also studied the read performance degradation due to flash reliability issues [24, 33, 41, 42]. Jung et al. observed the read latency increase caused by reliability management on reads, including read disturb mitigation, bad block management, and ECC [33]. Ha et al. proposed a read-disturb aware FTL to reduce the overhead of read reclamation, which mitigates read disturbance by migrating disturbed pages [24]. Liu et al. exploited the error locality in flash cells by caching bit-error information (not ECC redundancy) to accelerate the LDPC decoding [41, 42]. Our work is complementary to these works, since it targets a different scenario where the read performance degradation is caused by overlong ECC management.

Fig. 4. **Read hit rates of a data cache and two ECC caches.** These caches have the same capacity but different caching objects: data pages, ECC residues, and ECC pages. The ECC page cache has the potential to improve the read performance due to high read hit rates.

3.2 Potential of Caching ECC Residues

SSDs have built-in RAM to accommodate both a mapping cache and a data cache. The data cache accelerates user data access and reduces flash writes by caching active data pages. Different from coupling schemes, decoupling schemes separately store user data pages and their ECC residues. Hence, it is feasible to cache ECC residues, called an *ECC cache*. To retrieve a logical page in SSDs adopting an overlong ECC, zero or two flash page reads are needed, when the logical page hits or misses a data cache, respectively. By contrast, when a logical page hits the ECC cache, its ECC residue can be obtained in the cache and the page data and rest of its ECC unit need to be read from a flash page for decoding. That is, one or two flash page reads are required for an ECC cache hit or miss, respectively. Therefore, *an ECC cache has a higher hit overhead than a data cache*. However, an ECC residue is much smaller than a logical page, so an ECC cache can cover a larger working set than a data cache with the same capacity. Further, if we group ECC residues of logically consecutive data pages into ECC pages, caching ECC pages can exploit the spatial locality² that commonly exists in real-world workloads. Therefore, *an ECC cache achieves a higher hit rate than a data cache with the same capacity*.

Assume there are N logical page read requests. If the RAM is entirely used as a data cache with a hit rate H_{dc} or an ECC cache with a hit rate H_{ec} , then the number of flash page reads can be derived as Equation (1) or (2), respectively.

$$N_{dc} = (1 - H_{dc}) * 2 * N, \quad (1)$$

$$N_{ec} = H_{ec} * N + (1 - H_{ec}) * 2 * N. \quad (2)$$

When we expect the ECC cache causes fewer flash reads than the data cache, i.e., $N_{ec} < N_{dc}$, Equation (3) should be satisfied. This indicates that an ECC cache can reduce flash reads as long as achieving a two times higher hit rate, compared to a data cache. Figure 4 shows read hit rates of a data cache and two ECC caches, whose caching objects are individual ECC residues and ECC pages, respectively, with the same capacity under six real-world workloads. The sizes of a flash page, a logical page, and an ECC residue are 9, 8, and 0.5KB, respectively. An ECC page contains the ECC residues of 18 logically consecutive data pages. More detailed configurations can be found in Section 5.1. Compared to caching data pages, caching ECC pages and caching ECC residues achieve 73.2% and 32.2% higher read hit rates on average, respectively. Caching ECC pages satisfies Equation (3) under all the workloads, while caching ECC residues cannot. We can draw two conclusions from these results. First, an ECC cache has the potential to reduce flash reads in an

²Spatial locality refers to an access pattern: When a logical page is accessed, its logically neighboring pages are likely to be accessed soon.

Fig. 5. **Architecture of SCORE.** A fixed part of RAM is used as a mapping cache, while the other part is partitioned into a data cache and an ECC cache, between which the boundary is adjusted in a workload-adaptive manner. The ECC unit of a logical page is split into two parts: one stored in the data page and the other stored in an ECC page.

SSD adopting an overlong ECC. Second, the ECC cache should employ ECC pages, rather than individual ECC residues, as caching objects:

$$H_{ec} > 2 * H_{dc}. \quad (3)$$

These analyses motivate us to employ a hybrid cache design to partition available RAM into a data cache and an ECC cache. The sizes of two caches should be workload-adaptively adjusted, so the cache efficiency can be maximized under various workloads. Cache partitioning has been well studied in the CPU caches and storage systems [32, 49, 56, 65]. We extend this idea to the cache management inside an SSD adopting an overlong ECC. To our best knowledge, this is the first article that reveals the read performance problem and investigates the benefits of an ECC cache in an SSD adopting an overlong ECC.

4 DESIGN

We propose SCORE to efficiently cache overlong ECCs to improve the SSD performance. In this section, we first provide an overview of SCORE, and then we present the details about how to manage and cache overlong ECCs. Finally, we briefly discuss the data consistency issue.

4.1 Overview of SCORE

As shown in Figure 5, SCORE stores user data of a logical page and only part of its ECC unit in a data page, since the entire ECC unit cannot fit in the spare area. ECC residues of multiple data pages are grouped in one ECC page (where both the data area and spare area are utilized). Data pages are stored in data blocks, while ECC pages are stored in ECC blocks. A mapping table is maintained to locate data pages and ECC pages in flash memory. In this article, we assume a page-level mapping, because it has been widely used due to better addressing flexibility and performance [23]. The mapping table is organized into *translation pages* in an ascending order of *logical data page numbers (LPNs)*, which are stored in *translation blocks*. Note that MLC/TLC flash blocks can be operated in SLC mode or native MLC/TLC mode. SLC-mode blocks provide smaller access latencies and higher reliability at the cost of storage capacity loss [78]. SCORE

uniformly manages data blocks and ECC blocks using the native mode, because the volumes of user data and ECC residues are large. The mapping table is small and critical metadata that needs careful maintenance and fast accesses. Thus, some SLC-mode blocks are reserved as translation blocks [58]. A regular ECC, where ECC units are entirely stored in the spare area, is employed for translation pages. Such a multi-rate ECC design incurs a minimal hardware overhead [30].

Traditionally, RAM in an SSD serves as a data cache and a mapping cache, which accommodate user data pages and translation pages, respectively. SCORE adds an ECC cache whose caching objects are ECC pages by shrinking the data cache. In the current implementation, these caches adopt the *least recently used (LRU)* replacement policy with full associativity for simplicity and demonstration purpose (other advanced policies can also be adopted). On the critical path between caches and flash memory, the SSD controller employs an ECC engine to encode and decode data/translation pages. The ECC engine usually contains multiple pairs of encoders and decoders, as an SSD has multiple channels working in parallel.

When the SSD receives an I/O request, the FTL splits it into logical page requests. For each logical page request, SCORE checks whether it hits the data cache. If yes, then SCORE completes the page request by either returning the requested data to the host (for a read request) or updating the requested page in the data cache (for a write request). Regarding a read miss in the data cache, SCORE reads the requested data page from flash memory and checks whether the corresponding ECC residue hits the ECC cache. If an ECC cache miss occurs, then SCORE reads the corresponding ECC page from flash memory to the ECC cache. After the data page and its ECC unit are ready, the ECC engine decodes them. Then, the requested data can be obtained correctly and loaded to the data cache. Regarding a write miss in the data cache, SCORE places the new data page in the data cache. When a dirty data page is evicted in the data cache, it is encoded by the ECC engine. The raw data and a part of the ECC unit are written to a flash page. If the ECC residue hits the ECC cache, then the corresponding ECC page is updated in the cache. Otherwise, the ECC page is loaded to the ECC cache from flash memory first. When a dirty ECC page is evicted in the ECC cache, it is simply written to a flash page. During the whole procedure, address translation is performed through the mapping cache before reading/writing data/ECC pages in flash memory and garbage collection is triggered when the SSD runs out of free space.

4.2 Overlong ECC Management

How to group ECC residues into ECC pages determines the efficiency of accessing ECC pages. PG Decoupling groups ECC residues of physically consecutive data pages into ECC pages, as shown in Figure 2(b). This grouping is not efficient for two reasons. First, it minimally benefits from an ECC cache. Typically, written data involves randomness and is distributed over many channels/chips to exploit the internal parallelism of the SSD [34]. In PG Decoupling, ECC residues in an ECC page are not likely to be logically consecutive. Caching such ECC pages or individual ECC residues cannot exploit the spatial locality and thus becomes inefficient, as shown in Section 3.2. Second, PG Decoupling reserves a flash area to store ECC pages, complicating the flash management.

To avoid such drawbacks, SCORE employs the *LG Decoupling* scheme to manage overlong ECC, as shown in Figure 6. Multiple logically consecutive data pages form a *data group*, whose ECC residues are stored in an ECC page. An ECC page is labeled by a *data group number (DGN)*, which is the quotient of an LPN and the number of data pages in a data group. Mapping entries between DGNs and *physical page numbers (PPNs)* are used to locate ECC pages in flash memory. SCORE maintains a single mapping table by attaching DGN-PPN entries to the LPN-PPN table. Assuming the flash page size and ECC residue size are 9 and 0.5KB (the ECC code rate is 0.84), each data group contains 18 data pages. The mapping table size increases only by 1/18, compared to the

Fig. 6. **Logical View of LG Decoupling.** Suppose the flash page size and ECC residue size are 9 and 0.5KB (the ECC code rate is 0.84). ECC residues of 18 logically consecutive data pages form one ECC page. ECC pages are stored in ECC blocks, which are uniformly managed with data blocks.

conventional LPN-PPN table. Thus, only a minimum overhead is added to perform the address translation.

ECC residues in flash memory are accessed in the unit of an ECC page, which is marked as entirely valid or invalid. Thus, ECC blocks and data blocks can be uniformly managed with the same garbage collection and wear leveling algorithms. Although each ECC page contains logically consecutive ECC residues, SCORE does not require user writes to be sequential. An update of any ECC residue could lead to one ECC page update in the worst case, i.e., under small random writes, but this case is not common. Because write requests in real-world workloads usually exhibit good locality. A considerable number of residue updates can be absorbed by the ECC cache and residues that belong to the same ECC page can be written to flash memory in a batch, as shown in Section 5.2.

4.3 Overlong ECC Caching

As discussed in Section 3.2, an ECC cache is able to exploit the spatial locality and cover a larger working set, but it has a higher hit overhead, compared to a data cache. Note that workloads exhibit diverse access patterns. To take advantages of both caching schemes, SCORE partitions the cache into a data cache and an ECC cache and dynamically adjusts their sizes in a workload-adaptive fashion. We first present the hybrid cache design and then the cost models for caching efficiency evaluation.

4.3.1 Hybrid Cache Design. SCORE adopts the *ghost cache* idea [49, 56] to enable the workload-adaptive feature. Ghost caches track misses under different cache configurations (such as partition sizes [49] and number of ways [56]) at a low cost of RAM space and thus can provide trial experience for a better cache configuration. SCORE leverages ghost caches to derive the caching efficiency under different data cache and ECC cache sizes. Specifically, SCORE maintains three pairs of caches and each pair consists of a data cache and an ECC cache, as shown in Figure 7. The pair of caches, whose caching objects are data/ECC pages, are called *real caches* and their total size is fixed. The other two pairs of caches, called *ghost caches*, maintain only LPNs or DGNs, not the real data. Each pair of ghost caches simulate the same total size of real caches with a minimal space overhead, e.g., roughly 0.5% of the total cache space when the logical page size is 8KB. A pair of ghost caches, called *LD ghost caches*, always have a larger data cache than real data cache (and thus a smaller ECC cache than real ECC cache), while the other pair, called *SD ghost caches*, always have a smaller data cache than real data cache (and thus a larger ECC cache than real ECC cache).

Ghost caches provide trial experience for adjusting the sizes of real caches step-by-step periodically. When a period ends, SCORE evaluates which pair of caches perform the best using time cost models (presented in Section 4.3.2). If real caches perform the best, then all the cache sizes remain the same in the next period. If LD/SD ghost caches perform the best, then real data cache

Fig. 7. **Real caches with two pair of ghost caches.** SD/LD ghost caches, which accommodate only LPNs/DGNs (not actual data), always simulate a smaller/larger data cache than real data cache. Ghost caches provide trial experiences for adjusting the sizes of real caches step-by-step periodically.

and real ECC cache will be enlarged/shrunk and shrunk/enlarged by a step size, respectively, in the next period. Then, the sizes of ghost caches are adjusted accordingly. When a real cache is shrunk, SCORE reduces its size gradually instead of at once to avoid burst writes caused by dirty page evictions. Hence, cache size adjustments cause a negligible overhead. In the current implementation, we empirically set a period as the virtual time to process 10,000 page requests and the step size as 1/16 of the total size of real caches, as analyzed in Section 5.3.2.

4.3.2 Cost Models of Caching. To evaluate the caching efficiency, we develop time cost models for the three pair of caches, i.e., *real*, *sd*, and *ld*. Each time cost model includes three parts (as shown in Equation (4)): *IO cost* due to accessing user data and ECC redundancy in flash memory to serve I/O requests, *address translation (AT) cost* of accessing the mapping table in flash memory, *garbage collection (GC) cost* of reclaiming invalid flash pages when free blocks run out. Misses and writebacks of dirty data/ECC pages in the hybrid cache directly lead to the IO cost, which probably further causes AT and GC costs. A lower time cost indicates higher caching efficiency:

$$T = T_{IO} + T_{AT} + T_{GC}. \quad (4)$$

To calculate the time cost, we mainly consider seven types of operation latencies: *flash operation latencies* (i.e., SLC/native-mode page read/write and block erase), *data transfer latency* between flash memory and the SSD controller, and *ECC decoding latency*. The ECC encoding latency is negligible so as not to be considered. We refer to $\{op1, \dots, op7\}$ as the set of these types of operations (e.g., *op1* represents the SLC flash page read). Their latencies can be obtained from the datasheet or measured by the SSD controller.

Costs of Real Caches. SCORE periodically counts the numbers of operations in the set. Assume $N_{x(i)}^{real}$ is the number of a type of operations (indicated by i) caused by IO or AT or GC (indicated by x) and t_i is the latency of i operation. The IO/AT/GC cost can be calculated as Equation (5):

$$T_x^{real} = \sum_{i=op1}^{\{op1, \dots, op7\}} N_{x(i)}^{real} * t_i \quad (x = IO \text{ or } AT \text{ or } GC). \quad (5)$$

Costs of Ghost Caches. SCORE periodically derives the numbers of operations caused by IO according to the numbers of misses and writebacks in ghost caches. Thus, the *IO cost of SD/LD ghost caches* can be calculated in the same way as the IO cost of real caches. However, the ghost cache simulation does not include the mapping cache management and flash storage management. We cannot obtain the accurate AT and GC costs of ghost caches. Note that the sizes of ghost data/ECC caches are close to the sizes of real data/ECC caches. An approximate solution is to assume the average AT cost per IO page access and GC cost per IO page write of SD/LD ghost caches are equal to those of real caches. Assume the derived numbers of IO page accesses (including reads

and writes) and writes for each pair of caches are $N_{IO(rw)}^{sd/ld/real}$ and $N_{IO(w)}^{sd/ld/real}$. The AT and GC costs of SD/LD ghost caches can be estimated as Equation (6):

$$T_{AT}^{sd/ld} = N_{IO(rw)}^{sd/ld} * \frac{T_{AT}^{real}}{N_{IO(rw)}^{real}}, \quad T_{GC}^{sd/ld} = N_{IO(w)}^{sd/ld} * \frac{T_{GC}^{real}}{N_{IO(w)}^{real}}. \quad (6)$$

4.4 Discussion

A concern of SCORE is how to ensure the data consistency after a power failure, if volatile RAM is used for buffering dirty data. Both hardware and software solutions can be applied. From the hardware aspect, SSDs can employ non-volatile RAM (NVRAM), which ensures durability or volatile RAM with supercapacitors, which can flush dirty data to flash memory after a power failure [57]. SCORE gives a higher priority to dirty ECC than dirty data for flushing. This is because dirty ECC residues are generated only after dirty data pages are encoded when written to flash memory. If dirty ECC residues get lost, then the corresponding data pages in flash memory can no longer be correctly retrieved. By contrast, if dirty data pages get lost, the integrity of data pages that have been written to flash memory is not damaged. From the software aspect, dirty data can be flushed to flash memory periodically or conditionally when their volume exceeds the hardware protection capability. Furthermore, the FTL writes some metadata to the spare area of a flash page. The relevant LPN/DGN is written to the spare area of a data/ECC page. If the mapping table is corrupted, then it can be recovered by scanning the spare area of flash pages [48]. In our current implementation, we assume the caches employ NVRAM or DRAM with large supercapacitors to ensure the data consistency. We also show the experimental results in Section 5.3.4 where cache flushing is performed to limit the volume of dirty data assuming the hardware protection capability is very limited.

5 EVALUATION

5.1 Experiment Setup

We use trace-driven simulations to evaluate SCORE and compare it with Cross-Page [66], PG Decoupling [17], and Optimal. Cross-Page is regarded as the baseline to show performance improvements achieved by SCORE, as it performs better than PG Decoupling. Optimal, which is the ideal case with a large enough spare area, indicates the overhead of adopting an overlong ECC in a realistic SSD. These four schemes adopt the page-level mapping, where a logical page can be mapped to any physical page, and the greedy garbage collection policy, which selects the flash block with the least valid data as a victim to reclaim. They also reserve the same number of SLC-mode flash blocks as translation blocks.

The simulator is obtained by adding the four schemes into the Flashsim platform [23]. Before each simulation using a trace, we age the SSD by first sequentially writing the full disk and then writing it with the modified trace, where both reads and writes are changed to writes. This aging state helps to obtain more practical results. In addition, Flashsim does not consider the RAM access latency and cannot simulate the algorithm computation latency. Thus, we added them into the end-to-end latency model. The RAM access latency is calculated according to the DRAM latency model in MQsim [62, 63]. The average computation latency of processing an I/O request is estimated by simulating the execution of each scheme on the SimpleScalar-ARM platform [5], similar to estimate the delta-encoding latency in Reference [69]. The estimated latencies are 14μs for SCORE, 7μs for Cross-Page and PG Decoupling, and 6μs for Optimal.

In our simulations, SSD and ECC parameters are taken from References [4, 7] and listed in Table 1. The SSD user-visible capacity is configured to cover all or the majority (more than 80%) of

Table 1. SSD and ECC Configurations [4, 7]

Flash Page Size	8KB data area + 1KB spare area
Flash Block Size	1,152 pages (TLC), 384 flash pages (SLC)
Page Read Time	90 μ s (TLC), 50 μ s (SLC)
Page Program Time	1,100 μ s (TLC), 400 μ s (SLC)
Block Erase Time	10ms
Data Transfer Rate	400MB/s
ECC Decoding Time	20 μ s (TLC pages), 4 μ s (SLC pages)

Table 2. Specification of Workloads

	Trace Footprint	Working Set	Request (#)	Read Request Ratio	Avg. Request Size	
					Read	Write
wdev0	16GB	0.5GB	1,143,261	20.1%	12.6KB	8.2KB
usr0	16GB	2.5GB	2,236,836	40.4%	40.9KB	10.3KB
rsrch2	16GB	0.5GB	163,073	60.2%	4KB	4.3KB
mds1	256GB	72.9GB	1,315,759	93.8%	62KB	12.3KB
proj3	256GB	5.8GB	2,244,644	94.8%	9KB	23.7KB
web	16GB	5.9GB	4,182,047	100%	14.9KB	—

I/O requests in each trace, i.e., 16GB with 4 channels or 256GB with 8 channels in our experiments.³ All the four schemes benefit from the multi-channel architecture, since logical pages are distributed across the channels. The over-provisioning space ratio is configured as 25% [31]. The RAM space size is configured as 8 or 128MB in the 16 or 256GB SSD, respectively. A half of RAM space is used as the mapping cache, while the rest half is used for the hybrid cache in SCORE or a data cache in the other schemes. The default ECC code rate is 0.84, so an ECC page holds 18 ECC residues of 0.5KB each.

We choose six real-world traces to evaluate the four schemes. The *web* trace was collected from a popular search engine [1], while the other five traces were collected in different servers at Microsoft Research Cambridge (MSR) [2]. These traces have various read ratios, ranging from 20% to almost 100%, and request sizes. Table 2 presents the features of these workloads.

In the following subsections, we first provide the experimental results and analyses about performance, indicated by the average response time per I/O request. Then, we study the adaptivity of SCORE and sensitivities of several parameters. Finally, we present a write optimization technique for SCORE.

5.2 Performance Evaluation

5.2.1 System Response Time. Figures 8(a) and 8(b) show the average system and read response time of the four schemes under the six workloads. Cross-Page, PG Decoupling, and SCORE increase the average system response time by 51.1%, 59.7%, and 28.9%, respectively, compared to Optimal. These results show that adopting an overlong ECC results in a significant performance degradation. Compared to Cross-Page and PG Decoupling, SCORE decreases system response time

³Regarding the trace whose footprint is larger than the capacity, we truncate the trace footprint instead of mapping the large footprint onto the small capacity. We do *not* aggressively enlarge the SSD capacity, because it would result in an unrealistically large over-provisioning space.

Fig. 8. Average system response time (considering both read and write requests), average and 99th percentile read response time (considering only read requests).

Fig. 9. **Read performance analysis.** *Cross-Page-dp9* and *Cross-Page-dp18* refer to variants of *Cross-Page*, which perform data prefetching with lengths of 9 and 18 logical pages, respectively.

by an average of 14% and 18.2% and read response time by an average of 22.1% and 24.3%, respectively. In addition, SCORE reduces the 99th percentile read response time by an average of 9.9%, as shown in Figure 8(c). We note that SCORE has lower read response time under all the workloads, especially under read-dominant workloads. *These results demonstrate that SCORE achieves significant read performance improvements under various workloads, compared to existing schemes.* We also note that the system response time of SCORE is a little higher (smaller than 4%) than that of Cross-Page under write-dominant workloads (wdev0 and usr0). These results indicate SCORE has lower write efficiency, which is illustrated in Section 5.2.3.

5.2.2 Read Performance Analysis. Figure 9 provides a deeper insight into the read efficiency of the four schemes. With the same RAM capacity, SCORE maintains a data cache and an ECC cache, while the other schemes cache only data and thus have the same hit rates. Figure 9(a) shows read hit rates of a pure data cache without prefetching range from 0% to 49% with an average of 19.5%. These rates are low, because read requests in the workloads have relatively large working sets and weak temporal locality.⁴ Note that an ECC cache can cover a larger working set and exploit the spatial locality. SCORE achieves high ECC read hit rates, ranging from 47% to 90% with an average of 72.3%, at the cost of an average of 0.3% data hit rate degradation. These results indicate that SCORE strikes a good balance between caching data pages and ECC pages under various workloads. Hence, SCORE performs only one data page read in flash memory most of the time, when a logical page request misses the data cache. By contrast, Cross-Page needs two data page reads and PG Decoupling needs one data page read and one ECC page read.

Flash page reads come from I/O requests (i.e., reads and partial updates of data/ECC pages), address translation, and garbage collection. In the experiments, only a small fraction of flash page reads are caused by address translation and garbage collection. The mapping cache hit rates are

⁴Temporal locality refers to an access pattern: When a logical page is accessed, it is likely to be accessed again in the near future.

Fig. 10. Write performance analysis under real workloads. Only three workloads with considerable write requests (wdev0, usr0, rsrch2) are included.

larger than 99%, because the cache capacity is large (but reasonable) and caching translation pages exploits the spatial locality effectively. This also indicates that SCORE introduces a negligible address translation overhead in spite of extra DGN-PPN entries (as large as 1/18 of a pure LPN-PPN table). Garbage collection operations seldom occur, because the workloads have good write locality and/or small write volumes. Therefore, the read amplification mainly comes from I/O requests. As shown in Figure 9(b), the overall read amplifications of Cross-Page and PG Decoupling range from 2 to 2.7, while from 1 to 1.3 for Optimal. SCORE reduces the read amplification by an average of 74.3% or 75.6%, compared to both Cross-Page and PG Decoupling, while increases by an average of only 13.3%, compared to Optimal. These results demonstrate SCORE's high read efficiency.

Data prefetching is also able to exploit the spatial locality. To estimate whether it can achieve as high read performance as ECC prefetching, we evaluate two variants of Cross-Page, called *Cross-Page-dp9* and *Cross-Page-dp18*. They perform data prefetching with lengths of 9 and 18 logical pages, respectively. We can see data prefetching achieves high read cache hit rates ranging from 67% to 95% in Figure 9(a). It also better exploits the parallelism inside an SSD. However, such benefits are at the cost of high prefetching overheads and it is hard to determine an optimal prefetching length in various workloads. Prefetching logical data pages requires multiple flash page reads, while fetching 18 ECC residues in an ECC page requires only one flash page read. As shown in Figures 9(b) and 9(c), data prefetching significantly increases the read amplification and may degrade the read performance (by up to 3 $\times$). Therefore, by leveraging the page access feature of flash memory and small size of ECC residues, SCORE's ECC cache efficiently exploits the spatial locality with a minimum prefetching overhead. Although it is possible to enable both data and ECC prefetchings for higher read efficiency, we focus on exploiting the benefits of an ECC cache in this article.

5.2.3 Write Performance Analysis. We analyze write performance in this section. Only three workloads with considerable write requests are included, i.e., wdev0, usr0, and rsrch2. Figure 10(a) shows the average write response time of the four schemes. Compared to Cross-Page, SCORE degrades the write performance by an average of 13.1%. This write inefficiency is because SCORE couples an ECC page with multiple logically consecutive data pages. Any data page update could lead to one ECC page update. The ECC cache can effectively reduce ECC page updates under sequential writes, but random writes would result in extensive partial updates of ECC pages. To verify this, we studied the average ratio between the number of updated ECC residues in an ECC page write and the number of ECC residues that an ECC page contains, as shown in Figure 10(b). The average updating ratio is 26.5% and the lowest updating ratio is 10.5% in rsrch2 workload (because its write requests are the smallest). In addition, we also studied the write hit rates of a pure data cache and the hybrid cache of SCORE, as shown in Figure 10(c). Write hit rates of the data cache range from 63% to 69% with an average of 65.5%, indicating write requests in the three workloads have good temporal locality. The hybrid cache decreases data write hit rates by

Fig. 11. **Performance under synthetic workloads with poor spatial locality.** In the four read-dominant workloads, 100% (*r100*) or 70% (*r70*) of I/O requests are 4KB reads, where 50% (*s50*) or 10% (*s10*) are sequential while the other are random, and the rest are 4KB random writes. In the four write-dominant workloads, 100% (*w100*) or 70% (*w70*) of I/O requests are 4KB writes, where 50% (*s50*) or 10% (*s10*) are sequential while the other are random, and the rest are 4KB random reads. Each workload has two million I/O requests covering the entire logical space of the SSD.

an average of 1% and obtains an average of 29.6% ECC write hit rates. This benefit brought by the ECC cache is not large enough to compensate the write inefficiency.

Figure 10(d) shows the *write amplifications* (WAs)⁵ of the four schemes under the three workloads. The WAs range from 0.5 to 1, because many write requests are absorbed by the caches. Compared to Cross-Page and PG Decoupling, SCORE increases the write amplification by an average of 0.14 (19.9%) and 0.06 (9.1%), respectively. PG Decoupling performs worse than Cross-Page and even SCORE due to higher accumulated garbage collection overheads in the data area and ECC area. More flash page writes lead to more flash block erases and thus worse SSD lifetime. Specifically, SCORE has an average of 11.3% and 0.9% higher numbers of block erases than Cross-Page and PG Decoupling, respectively (we do not show this figure in the article). These results show SCORE is not efficient to handle small random writes.

5.2.4 Synthetic Workloads. In this section, we evaluate the four schemes using synthetic workloads with poor spatial locality and large working sets. As shown in Figures 11, the average read and write response time of the four schemes increases as the I/O sequentiality decreases from 50% to 10%. Such degradations are due to lower cache hit rates, higher address translation and garbage collection overheads. Note that the sizes of flash pages and I/O requests are 8 and 4KB, respectively. Caching a data page enables prefetching of 4KB data, so random requests result in non-zero data cache hit rates (except in SCORE under *r70-s10* workload). In read-dominant workloads, SCORE averagely achieves 12.6% better read performance than Cross-Page. In write-dominant workloads, SCORE performs worse than Cross-Page, e.g., 56.2% in write performance and 80.5% in write amplification, on average. These results show SCORE's advantage on read efficiency even under poor spatial locality but also its inefficiency under small random writes.

It is important to note that SCORE aims at improving the read performance. On the one hand, SSDs are widely used to serve read-critical workloads with limited random writes [19, 40, 55, 60, 72, 75] and many popular applications and file systems are designed to transform random writes to sequential writes (such as log-structured file systems and key-value stores) [37, 43, 52]. In these scenarios, SCORE is able to significantly improve the read performance with a small write overhead. On the other hand, it is feasible to enhance SCORE by integrating Cross-Page to manage randomly written data. As shown in Section 5.4, such an enhancement achieves both high read efficiency and high write efficiency.

⁵Write amplification, which refers to the ratio of the volume of flash writes to the volume of user writes, reflects the SSD write efficiency.

Fig. 12. Ratio of real data cache size to total real cache size over time in SCORE. The time is indicated by the number of processed logical page requests.

5.3 Sensitivity Study of SCORE

5.3.1 Adaptivity of SCORE. As presented in above sections, the data cache and ECC cache show different efficiencies under different access patterns. SCORE workload-adaptively adjusts the data cache and ECC cache sizes to gradually approach the size configuration that maximizes the caching efficiency, as illustrated in Section 4.3. Figure 12 shows the ratios of data cache size to total cache size over time under the six workloads. The time is indicated by the number of processed logical page requests in each workload. We can see the data cache size ratios are significantly different under different workloads, ranging from 0% to 95%. The ratios are zero most of the time in web workload, i.e., the whole RAM is used as an ECC cache. This is because web workload has a large working set and good spatial locality. In the other workloads, the ratios are larger than 50% most of the time, which indicates a relatively small ECC cache is enough to obtain high hit rates. Furthermore, the data cache size is likely to change over time in a workload. For example, the ratios ranges from 12.5% to 93.8% in usr0 workload and from 12.5% to 68.8% in proj3 workload during the whole running time (Figure 12 shows only a part of the running time for demonstration purpose). We can draw two conclusions from these results. First, access patterns in different workloads and in different periods of a workload can be diverse. Thus, a hybrid cache with fixed cache partitions cannot maximize the caching efficiency. Second, the hybrid cache design is adaptive to diverse workloads and achieves good performance.

5.3.2 Selection of Period and Step Sizes. SCORE periodically adjusts the data cache size and ECC cache size in a step unit. A longer period provides more history information for the cost models, but reduces the sensitivity to access pattern changes. A smaller step indicates more fine-grained adjustments, but a larger possibility to achieve a local optimum. Considering diverse workloads, it is hard to derive an optimal period length and step size. It is also not possible to verify all the choices. Thus, we studied the average system response time of SCORE under some typical period lengths and step sizes, as shown in Figure 13. We can see the performance of SCORE is not sensitive to the period lengths and step sizes under most workloads, and a moderate period length or step size results in relatively good performance. Therefore, we set the period length as the logical time to process 10,000 page requests and the step size as 1/16 of the total size of real caches in the current implementation.

Fig. 13. Selections of *period* length and *step* size in SCORE.

Fig. 14. Average system response time of SCORE (with a hybrid cache) and Cross-Page (with a data cache) under different RAM sizes.

5.3.3 Impacts of RAM Sizes. In this section, we investigate the impacts of RAM sizes on the system response time of SCORE and Cross-Page, as shown in Figure 14. PG Decoupling is not included due to its worse performance than Cross-Page. We can see the performance of both SCORE and Cross-Page improves under wdev0 and usr0 workloads but remains almost the same under the other workloads as the RAM size increases. These results show that a small RAM is effective enough in wdev0 and usr0 workloads. Figure 14(c) shows the performance improvement of SCORE over Cross-Page tends to be slightly larger as the RAM size increases. This is because SCORE can utilize the RAM space more efficiently.

5.3.4 Impacts of Cache Flushing. In this section, we assume the SSD employs DRAM with very limited hardware protection capability, e.g., a small capacitor. To ensure data consistency after a power failure, cache flushing is necessary to limit the volume of dirty data. Figure 15 shows the average system and read response time, read and write amplifications of the four schemes enabling cache flushing, where the volume of dirty data is kept below 30% or 10% of the RAM size. Only three workloads with considerable writes (wdev0, usr0, rsrch2) are included, because cache flushing has negligible impacts in the other workloads. We can see enabling cache flushing introduces noticeable overheads in the four schemes, e.g., the system response time increases by 3% to 5% or 18% to 31% (compared to Figure 8(a)) and the write amplification increases by 3% to 6% or 17% to 39% (compared to Figure 10(d)) under a threshold of 30% or 10%, respectively. Compared to Cross-Page, SCORE has similar or 7.3% higher system response time and 22.9% or 40.3% higher write amplification under a threshold of 30% or 10%, respectively, on average. Such degradations are due to SCORE's write inefficiency. However, SCORE still achieves an average of 40% or 36.3% lower read amplification and 16.4% or 20.9% lower read response time under a threshold of 30% or 10%, respectively. The reason why more aggressive cache flushing leads to better read performance

Fig. 15. Impacts of cache flushing on average system response time and read response time, read amplification (RA), and write amplification (WA). The volume of dirty data is limited below 30% or 10% of the RAM size. Only three workloads with considerable writes (wdev0, usr0, rsrch2) are included.

Fig. 16. Read cache hit rate, read, and write performance under an ECC code rate of 0.8. Only three workloads with considerable write requests are included in Figure (c).

is that a smaller volume of dirty data leads to smaller read and write interference. These results show that SCORE retains its advantage on read performance when cache flushing is enabled.

5.3.5 Impacts of ECC Code Rates. The above experimental results are based on an ECC code rate of 0.84. In this section, we investigate the impact of ECC code rates on the performance of the four schemes. With a lower code rate, the ECC unit size becomes larger and thus both read amplification and write amplification increase. Moreover, each ECC page contains fewer ECC residues, so the ECC cache hit rate decreases. Figure 16 shows the read cache ratios, average read and write response time of the four schemes under an ECC *code rate* (*CR*) of 0.8. The ECC residue size is 1KB and each ECC page contains 9 ECC residues. Other experimental configurations are set by default. We can see the ECC read hit rates of SCORE are still high, ranging from 44% to 84% with an average of 66.7% (while 72.3% under 0.84 CR). Accordingly, compared to Cross-Page and PG Decoupling, SCORE reduces the average read response time by an average of 16.8% and 21.1% (while 22.1% and 24.3% under 0.84 CR) under all the workloads, respectively. Therefore, SCORE still maintains a considerable read performance improvement, as the ECC code rate decreases. Regarding the write performance, only three workloads with considerable write requests are included, i.e., wdev0, usr0, and rsrch2. SCORE increases the average write response time by an average of 6.6% (while 13.1% under 0.84 CR), compared to Cross-Page. This write performance degradation becomes smaller, because each ECC page is coupled to fewer logical data pages. In addition, SCORE has 56.9% (while

Fig. 17. Write optimization of SCORE. The experimental configurations are set by default.

28.9% under 0.84 CR) higher system response time than Optimal. Hence, lowering the code rate dramatically increases the overhead of adopting an overlong ECC. It should be prudent to choose an extremely low code rate.

5.4 Write Optimization

SCORE is efficient in handling read requests and sequential write requests, but not random write requests, as analyzed in Section 5.2.3. We note that most user data are read dominant or write dominant [39]. To reduce the write overhead, a potential optimization is to use SCORE to manage data that is read-dominant or sequentially written and a complementary scheme to manage randomly written data. We have implemented such an optimization based on SCORE, called *SCORE-plus*, which employs Cross-Page as the complementary scheme. This is because Cross-Page can efficiently handle random writes and be easily integrated into SCORE. Figure 17(a) shows the write amplifications of Cross-Page, SCORE, and SCORE-plus under wdev0, usr0, and rsrch2 workloads. Compared to Cross-Page, SCORE-plus increases the write amplification by only an average of 0.06 or 8.2% (while 0.14 or 19.9% for SCORE). Meantime, SCORE-plus offers similar performance advantages over Cross-Page, compared to SCORE, as shown in Figure 17(b). These results demonstrate that SCORE-plus largely eliminates the write overhead without sacrificing the performance.

6 CONCLUSION

As flash technologies scale aggressively, degrading reliability and lifetime have become a critical concern. Some SSDs adopt overlong ECCs to solve this concern at the cost of a large read performance degradation. In this article, we propose SCORE to improve the SSD performance by caching the overlong ECC. Since caching ECC achieves a higher hit rate but higher hit overhead than caching user data, SCORE employs a hybrid cache that accommodates both user data and overlong ECC in a workload-adaptive manner. Experimental results show SCORE obtains high ECC hit rates without sacrificing data hit rates, and thus significantly improves the SSD performance by reducing flash reads. Furthermore, SCORE-plus, which refers to SCORE with an optimization towards random writes, maintains the performance advantage with a marginal write overhead. We believe that SCORE delivers an effective solution to the read performance problem of SSDs adopting overlong ECCs.

ACKNOWLEDGMENTS

The authors are grateful to the editors and reviewers for their insightful comments that have greatly improved the article.

REFERENCES

- [1] I/O Traces. 2007. Traces from UMass Trace Repository. Retrieved from <http://traces.cs.umass.edu/index.php/Storage/Storage>.
- [2] I/O Traces. 2011. MSR Cambridge Traces. Retrieved from <http://iota.snia.org/traces/388>.
- [3] JESD218. 2016. The JEDEC Standard of Solid State Drives. Retrieved from <https://www.jedec.org/standards-documents/focus/flash/solid-state-drives>.
- [4] SK Hynix. 2016. SK Hynix V3 256Gb TLC NAND Flash Memory Data Sheet.
- [5] Todd Austin and Doug Burger. 2000. SimpleScalar-ARM. Retrieved from <http://simplescalar.com/v4test.html>.
- [6] Stephen Bates. 2013. Using rate-adaptive LDPC codes to maximize the capacity of SSDs. *Proceedings of the Flash Memory Summit*.
- [7] Stephen Bates. 2014. Latency in LDPC-based next-generation SSD controllers. Retrieved from <http://blog.pmcs.com/latency-in-ldpc-based-next-generation-ssd-controllers/>.
- [8] 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 (2017), 1666–1704.
- [9] Yu Cai, Saugata Ghose, Yixin Luo, Ken Mai, Onur Mutlu, and Erich F. Haratsch. 2017. Vulnerabilities in MLC NAND flash memory programming: Experimental analysis, exploits, and mitigation techniques. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'17)*.
- [10] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. 2012. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE'12)*.
- [11] Yu Cai, Yixin Luo, Saugata Ghose, and Onur Mutlu. 2015. Read disturb errors in MLC NAND flash memory: Characterization, mitigation, and recovery. In *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'15)*.
- [12] Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. 2015. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA'15)*.
- [13] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. 2013. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *Proceedings of the 31st IEEE International Conference on Computer Design (ICCD'13)*.
- [14] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Cristal, Osman S. Unsal, and Ken Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Proceedings of the 30th IEEE International Conference on Computer Design (ICCD'12)*.
- [15] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Osman Unsal, Adrian Cristal, and Ken Mai. 2014. Neighbor-cell assisted error correction for MLC NAND flash memories. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'14)*.
- [16] Li-Pin Chang. 2007. On efficient wear leveling for large-scale flash-memory storage systems. In *Proceedings of ACM Symposium on Applied Computing*.
- [17] Yuan-Hao Chang, Ming-Chang Yang, Tei-Wei Kuo, and Ren-Hung Hwang. 2013. A reliability enhancement design under the flash translation layer for MLC-based flash-memory storage systems. *ACM Trans. Embed. Comput. Syst.* 13, 1 (2013), 10:1–10:28.
- [18] Hyojin Choi, Wei Liu, and Wonyong Sung. 2010. VLSI implementation of BCH error correction for multilevel cell NAND flash memory. *IEEE Trans. Very Large Scale Integr. Syst.* 18, 5 (2010), 843–847.
- [19] Bob Fine. 2016. McKesson mixes SSDs with HDDs for Optimal Performance and ROI. In *Proceedings of the Flash Memory Summit*.
- [20] Eran Gal and Sivan Toledo. 2005. Algorithms and data structures for flash memories. *ACM Comput. Surveys* 37, 2 (2005), 138–163.
- [21] Peter Graumann. 2016. LDPC code rate adaptation methods for NAND flash. *Proceedings of the Flash Memory Summit*.
- [22] Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*.
- [23] Aayush Gupta, Youngjae Kim, and Bhuvan Ugaonkar. 2009. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS'09)*.
- [24] Keonsoo Ha, Jaeyong Jeong, and Jihong Kim. 2016. An integrated approach for managing read disturbs in high-density NAND flash memory. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* 35, 7 (2016), 1079–1091.
- [25] Jim Handy. 2017. Flash market current and future. *Proceedings of the Flash Memory Summit*.
- [26] Erich F. Haratsch. 2016. NAND flash media management algorithms. *Proceedings of the Flash Memory Summit*.

- [27] Chien-Chung Ho, Yuan-Hao Chang, Che-Wei Tsao, and Pei-Lun Sui. 2014. Block reinforcement to optimize lifetime of flash storage devices. In *Proceedings of the 3rd IEEE Global Conference on Consumer Electronics (GCCE'14)*.
- [28] Chien-Chung Ho, Yu-Ping Liu, Yuan-Hao Chang, and Tei-Wei Kuo. 2016. Antiwear leveling design for SSDs with hybrid ECC capability. *IEEE Trans. Very Large Scale Integr. Syst.* 25, 2 (2016), 488–501.
- [29] Jen-Wei Hsieh, Chung-Wei Chen, and Han-Yi Lin. 2015. Adaptive ECC scheme for hybrid SSD's. *IEEE Trans. Comput.* 64, 12 (2015), 3348–3361.
- [30] Xinde Hu. 2012. LDPC codes for flash channel. *Proceedings of the Flash Memory Summit*.
- [31] Ping Huang, Guanying Wu, Xubin He, and Weijun Xiao. 2014. An aggressive worn-out flash block management scheme to alleviate SSD performance degradation. In *Proceedings of the 9th ACM European Conference on Computer Systems (EuroSys'14)*.
- [32] Norman P. Jouppi. 1990. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA'90)*.
- [33] Myoungsoo Jung and Mahmut Kandemir. 2013. Revisiting widely held SSD expectations and rethinking system-level implications. In *Proceedings of ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'13)*.
- [34] Myoungsoo Jung and Mahmut T. Kandemir. 2014. Sprinkler: Maximizing resource utilization in many-chip solid state disks. In *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA'14)*.
- [35] Sanghyuk Jung, Sangyong Lee, Hoesung Jung, and Yong Ho Song. 2011. In-page management of error correction code for MLC flash storages. In *Proceedings of the 54th IEEE International Midwest Symposium on Circuits and Systems (MWSCAS'11)*.
- [36] Hyunsuk Kim, Su-Jin Ahn, Yu Gyun Shin, Kyupil Lee, and Eunseung Jung. 2017. Evolution of NAND flash memory: From 2D to 3D as a storage market leader. In *IEEE International Memory Workshop (IMW'17)*.
- [37] Changman Lee, Dongho Sim, Jooyoung Hwang, and Sangyeun Cho. 2015. F2FS: A new file system for flash storage. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST'15)*.
- [38] Hakyong Lee, Sanghyuk Jung, and Yong Ho Song. 2012. PCRAM-assisted ECC management for enhanced data reliability in flash storage systems. *IEEE Trans. Consum. Electron.* 58, 3 (2012), 849–856.
- [39] Qiao Li, Liang Shi, Chun Jason Xue, Kaijie Wu, Cheng Ji, Qingfeng Zhuge, and Edwin H.-M. Sha. 2016. Access characteristic guided read and write cost regulation for performance improvement on flash memory. In *Proceedings of the 14th Usenix Conference on File and Storage Technologies (FAST'16)*.
- [40] Ruixuan Li, Chengzhou Li, Weijun Xiao, Hai Jin, Heng He, Xiwu Gu, Kunmei Wen, and Zhiyong Xu. 2012. An efficient SSD-based hybrid storage architecture for large-scale search engines. In *Proceedings of the 41st IEEE International Conference on Parallel Processing*.
- [41] Ren-Shuo Liu, Meng-Yen Chuang, Chia-Lin Yang, Cheng-Hsuan Li, Kin-Chu Ho, and Hsiang-Pang Li. 2014. EC-cache: Exploiting error locality to optimize LDPC in NAND flash-based SSDs. In *Proceedings of the 51st ACM Design Automation Conference (DAC'14)*.
- [42] Ren-Shuo Liu, Meng-Yen Chuang, Chia-Lin Yang, Cheng-Hsuan Li, Kin-Chu Ho, and Hsiang-Pang Li. 2016. Improving read performance of NAND flash SSDs by exploiting error locality. *IEEE Trans. Comput.* 65, 4 (2016), 1090–1102.
- [43] Lanyue Lu, Thanumalayan Sankaranarayanan Pillai, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. WiscKey: Separating keys from values in SSD-conscious storage. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*.
- [44] Yixin Luo, Yu Cai, Saugata Ghose, Jongmoo Choi, and Onur Mutlu. 2015. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In *Proceedings of IEEE Symposium on Mass Storage Systems and Technologies (MSST'15)*.
- [45] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2016. Enabling accurate and practical online flash channel modeling for modern MLC NAND flash memory. *IEEE J. Select. Areas Commun.* 34, 9 (2016), 2294–2311.
- [46] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2018. HeatWatch: Improving 3D NAND flash memory device reliability by exploiting self-recovery and temperature awareness. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*.
- [47] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2018. Improving 3D NAND flash memory lifetime by tolerating early retention loss and process variation. In *Abstracts of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'18)*.
- [48] Dongzhe Ma, Jianhua Feng, and Guoliang Li. 2011. LazyFTL: A page-level flash translation layer optimized for NAND flash memory. In *Proceedings of ACM SIGMOD International Conference on Management of Data*.
- [49] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03)*.
- [50] Micron. 2015. MT29F384G08EBHBB 3D TLC NAND Flash Memory Data Sheet.

- [51] Neal R. Mielke, Robert E. Frickey, Ivan Kalastirsky, Minyan Quan, Dmitry Ustinov, and Venkatesh J. Vasudevan. 2017. Reliability of solid-state drives based on NAND flash memory. *Proc. IEEE* 105, 9 (2017), 1725–1750.
- [52] Changwoo Min, Kangnyeon Kim, Hyunjin Cho, Sang-Won Lee, and Young Ik Eom. 2012. SFS: Random write considered harmful in solid state drives. In *Proceedings of USENIX Conference on File and Storage Technologies (FAST'12)*.
- [53] Shigeo Ohshima and Yoichiro Tanaka. 2016. New 3D flash technologies offer both low cost and low power solutions. *Proceedings of the Flash Memory Summit*.
- [54] Michael Oros. 2018. Analysts weigh in on persistent memory. *Proceedings of the Persistent Memory Summit*.
- [55] Jian Ouyang, Shiding Lin, Song Jiang, Zhenyu Hou, Yong Wang, and Yuanzheng Wang. 2014. SDF: Software-defined flash for web-scale internet storage systems. In *Proceedings of the 19th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14)*.
- [56] Moinuddin K. Qureshi and Yale N. Patt. 2006. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*.
- [57] Samsung Electronics Co. 2014. Power loss protection (PLP): Protect your data against sudden power loss. Retrieved from http://www.samsung.com/semiconductor/minisite/ssd/downloads/document/Samsung_SSD_845DC_05_Power_loss_protection_PLP.pdf.
- [58] Deepak Sharma. 2014. System design for mainstream TLC SSD. *Proceedings of the Flash Memory Summit*.
- [59] Siva Sivaram. 2016. Storage class memory: Learning from 3D NAND. *Proceedings of the Flash Memory Summit*.
- [60] Yevgeniy Sverdlik. 2013. Open Compute: Facebook to move all archival data onto all-Flash storage. Retrieved from <http://www.datacenterdynamics.com/content-tracks/servers-storage/open-compute-facebook-to-move-all-archival-data-onto-all-flash-storage/72665.fullarticle>.
- [61] Shuhei Tanakamaru, Masafumi Doi, and Ken Takeuchi. 2013. Error-prediction analyses in 1X, 2X and 3Xnm NAND flash memories for system-level reliability improvement of solid-state drives (SSDs). In *Proceedings of the IEEE International Reliability Physics Symposium (IRPS'13)*.
- [62] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices. In *Proceedings of the 16th USENIX Conference on File and Storage Technologies (FAST'18)*.
- [63] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQsim GitHub Repository. Retrieved from <https://github.com/CMU-SAFARI/MQSim>.
- [64] Toshiba. 2015. TH58TET0THLBA8H TLC NAND Memory Toggle DDR2.0 Technical Data Sheet.
- [65] Chundong Wang and Wengfai Wong. 2013. TreeFTL: Efficient RAM management for high performance of NAND flash-based storage systems. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE'13)*.
- [66] Shunzhuo Wang, Fei Wu, Zhonghai Lu, You Zhou, Qin Xiong, Meng Zhang, and Changsheng Xie. 2017. Lifetime adaptive ECC in NAND flash page management. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE'17)*.
- [67] Mark Webb. 2017. 3D NAND status and roadmap 2017. *Proceedings of the Flash Memory Summit*.
- [68] Fei Wu, Yue Zhu, Qin Xiong, Zhonghai Lu, You Zhou, Weizhen Kong, and Changsheng Xie. 2018. Characterizing 3D charge trap NAND flash: Observations, analyses and applications. In *Proceedings of the 36th IEEE International Conference on Computer Design (ICCD'18)*.
- [69] Guanying Wu and Xubin He. 2012. Delta-FTL: Improving SSD lifetime via exploiting content locality. In *Proceedings of the 7th ACM European Conference on Computer Systems (EuroSys'12)*.
- [70] Guanying Wu and Xubin He. 2012. Reducing SSD read latency via NAND flash program and erase suspension. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST'12)*.
- [71] Guanying Wu, Xubin He, Ningde Xie, and Tong Zhang. 2013. Exploiting workload dynamics to improve SSD read latency via differentiated error correction codes. *ACM Trans. Design Automat. Electron. Syst.* 18, 4 (2013), 55:1–55:22.
- [72] Qianbin Xia and Weijun Xiao. 2015. Flash-aware high-performance and enduring cache. In *Proceedings of the 23rd IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'15)*.
- [73] Qin Xiong, Fei Wu, Zhonghai Lu, Yue Zhu, You Zhou, Yibing Chu, Changsheng Xie, and Ping Huang. 2018. Characterizing 3D floating gate NAND flash: Observations, analyses, and implications. *ACM Trans. Stor.* 14, 2 (2018), 16:1–16:31.
- [74] Shiqin Yan, Huaicheng Li, Mingzhe Hao, Michael Hao Tong, Swaminathan Sundararaman, Andrew A. Chien, and Haryadi S. Gunawi. 2017. Tiny-tail flash: Near-perfect elimination of garbage collection tail latencies in NAND SSDs. In *Proceedings of the 15th USENIX Conference on File and Storage Technologies (FAST'17)*.
- [75] Qing Yang and Jin Ren. 2011. I-CASH: Intelligently coupled array of SSD and HDD. In *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture (HPCA'11)*.

- [76] Jung H. Yoon, Ranjana Godse, Gary Tressler, and HilleryHunter. 2017. 3D-NAND scaling & 3D-SCM—Implications to Enterprise Storage. *Flash Memory Summit*.
- [77] Cristian Zambelli, Giuseppe Cancelliere, Fabrizio Riguzzi, Evelina Lamma, Piero Olivo, Alessia Marelli, and Rino Micheloni. 2017. Characterization of TLC 3D-NAND flash endurance through machine learning for LDPC code rate optimization. In *Proceedings of the IEEE International Memory Workshop (IMW'17)*.
- [78] Xuebin Zhang, Jiangpeng Li, Hao Wang, Kai Zhao, and Tong Zhang. 2016. Reducing solid-state storage device write stress through opportunistic in-place delta compression. In *Proceedings of the 14th USENIX Conference on File and Storage Technologies (FAST'16)*.
- [79] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. LDPC-in-SSD: Making advanced error correction codes work effectively in solid state drives. In *Proceedings of the 11th USENIX Conference on File and Storage Technologies (FAST'13)*.
- [80] You Zhou, Fei Wu, Ping Huang, Xubin He, Changsheng Xie, and Jian Zhou. 2015. An efficient page-level FTL to optimize address translation in flash memory. In *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys'15)*.

Received June 2018; revised October 2018; accepted November 2018