Improving MLC Flash Performance with Workload-Aware Differentiated ECC

Qianbin Xia and Weijun Xiao
Department of Electrical and Computer Engineering
Virginia Commonwealth University
Email: {xiaq2, wxiao}@vcu.edu

***Abstract*—The adoption of small geometries and multi-level cell (MLC) technologies significantly expands the capacity and drops the price of flash memory, which, at the same time, noticeably degrades the performance and reliability of the devices. As incremental-step pulse programming (ISPP) scheme is used to increase the programming accuracy for MLC cells, there is a trade-off between the SSD write performance and raw storage reliability. What's more, ECC is widely used in SSDs to provide error-tolerance ability. Therefore, if we could use stronger ECC to increase error correction strength, a low-cost write with coarser step sizes could be applied in the ISPP scheme to promote the write performance. However, stronger ECC scheme may hurt the read performance due to the increased decoding complexity and latency.**

In this paper, we propose a workload-aware differentiated ECC scheme to improve the SSD write performance without sacrificing the read performance. The main idea is to dynamically classify the logical pages into three categories: write-only, read-only, and overlapped part. For write-only logical pages, low-cost write with strong ECC scheme will be applied to increase the write performance. For write logical pages in the overlapped part, the low-cost writes with strong ECC will be selectively used based on their relative write and read hotness. While for any read logical pages encoded with a stronger ECC, we will rewrite them with the normal-cost write and ECC scheme if their hotness exceed a pre-defined threshold. The evaluation results show that our workload-aware differentiated ECC scheme could reduce the write and read response times by 48% and 11% on average, respectively. Even compared with the latest previous work, our workload-aware design can still gain about 4% write performance and 11% read performance improvements.

***Index Terms*—ISPP; BCH; LDPC; Bloom Filter;**

I. Introduction

NAND Flash-based Solid State Disk (SSD) has been widely deployed in various environments because of its high performance, nonvolatile property, and low power consumption. To continuously increase the capacity and reduce the bit cost, manufactures are aggressively scaling down the geometries and storing more bits information per flash cell [1]. However, the adoption of these technologies has inevitably resulted in degraded SSD performance especially the write performance. For example, from the previous SLC SSDs to current 2-bit MLC SSDs, the page write latency has increased from 200 us [39] to 1800 us [40], which has been identified as the major performance bottleneck [41].

A flash cell uses floating gate to store electrons and the amount of electrons will affect the cell's threshold voltage V_{th} [4]. Different V_{th} values could represent different data. These operations performed to inject and remove electrons from the floating gate are called program and erase, respectively [5]. Currently, the incremental-step pulse programming (ISPP) scheme [6] is used to perform the flash write operations. ISPP consists of a series of programming-and-verifying steps. In each step, a programming voltage $V_{program}$ is firstly applied to raise a cell's threshold voltage to $V_{current}$, then during the verification stage, the $V_{current}$ will be compared with the expected threshold voltage V_{expect}. If the $V_{current}$ is larger than the V_{expect}, then the program operation is complete. Otherwise, the $V_{program}$ will be increased by a step voltage ΔV_{pp} and the programming-and-verifying steps will be repeated. Therefore, increasing the step voltage ΔV_{pp} could reduce the programming steps and the overall time to reach the expected threshold voltage and improve the flash write performance. However, a larger ΔV_{pp} will result in worse raw bit error rate (RBER) due to the wider threshold voltage distribution of each programmed state and less noise margin between adjacent programmed states. To compensate the increased error rate due to a larger ΔV_{pp}, a stronger ECC scheme could be applied to provide stronger error correction capability. Currently, BCH code [10] is being widely used in NAND flash memories [7–9]. Beside BCH, the low-density parity-check (LDPC) codes [11, 12] have attracted a lot of attention in both academic and industrial communities [13, 15, 16] and are the promising substitutes of the BCH code for the future SSDs. Compared with BCH code, LDPC could provide superior error correction capability. However, the complicated and time-consuming decoding process of LDPC will inevitably worsen the flash read performance and limit its rapid adoption in the real SSD product. It has been showed in [16] that the LDPC code can increase the read latency by almost 120%, while the effects on the flash write performance is within 2%.

In this work, we propose a workload-aware differentiated ECC design to improve the write performance of 2-bit MLC flash devices. BCH code is used as our baseline ECC scheme, while LDPC is utilized as the stronger ECC to accelerate the write performance. First, we dynamically separate the read and write logical pages. For write-only pages, LDPC will be applied to improve flash write performance by increasing the step voltage ΔV_p. For write pages in the overlapped section, LDPC with low-cost write scheme will be selectively applied based on the relative write and read hotness of the logical pages. Since LDPC will dramatically degrade the

read performance, a rewrite operation with BCH and normal-cost write scheme will be performed on the logical pages whose read hotness exceed a pre-defined threshold. Our main contributions in this paper include:

- We propose a workload-aware differentiated ECC scheme to improve Flash write performance without compromising Flash read performance.
- We design and implement a lightweight read and write separator by modifying the basic multi-bloom filters, which could dynamically separate the read and write logical pages in both the spatial and temporal spaces.
- we present an efficient implementation of our design. Based on the simulations of realistic disk traces, we demonstrate that our proposed design could reduce the Flash write latency by 48% on average without sacrificing the read performance.

The rest of the paper is organized as follows. In Section II, we describe the background and related work on flash memory. Section III presents workloads analysis. Section IV depicts the details of our proposed workload-aware differentiated ECC scheme. Section V presents the evaluation methodology and the experimental results. Section VI presents concluding remarks.

II. Background and Related Work

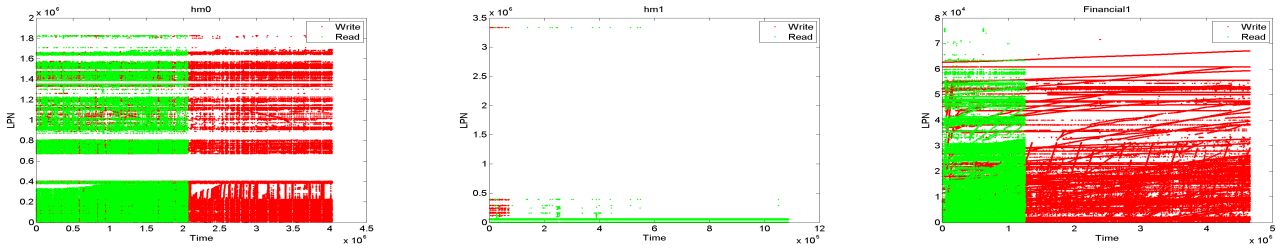
A. NAND Flash-Based SSD

In the commercial market, there are three widely used types of flash cells: SLC (single-level cell, storing a single bit per memory cell), MLC (multilevel cell, usually referred to two bits per cell), and TLC (Tri-level cell, storing three bits per cell). Storing multiple bits information in a cell could significantly improve the storage density and capacity with the penalty of sacrificed reliability, lifetime, and performance. In this paper, we only consider the two bits per cell MLC flash memory.

Unlike traditional storage devices, Flash memory can not support in-place update and has the lifetime limitation. In order to make flash memory compatible with the existing file systems designed for in-place update devices, an Flash Translation Layer (FTL) is deployed in SSDs. An typical FTL consists of an address translator, garbage collector, and wear-lever. The address translator translates the logical page number to the physical page number and maintains a mapping table that associates the logical page number with physical page number. There are three main mapping scheme: page-level mapping [17] that can achieve the best performance with the highest memory overhead, block-level mapping [18] that can save huge amount of memory with sacrificed performance, and hybrid mapping [19, 20] that makes a compromise between the page-level mapping and block-level mapping. A garbage collector reclaims the obsolete pages due to out-of-place updates. One of the most popular garbage collection schemes is the greedy algorithm [21], which always selects the blocks with the fewest number of valid pages as the victim blocks. The objective of a wear-leveler [22] is to evenly distribute the writes among all the flash blocks to improve the overall lifetime of flash memory. For simplicity, we assume that the page-level mapping scheme, greedy garbage collection policy, and none wear-leveler function unit are used in this paper.

B. Related Work

With the adoption of small geometrics and MLC technologies, the degraded performance of flash-based SSDs is becoming a critical issue. Many schemes have been proposed to enhance the performance. Some of the previous research work is based on the trade-off between the RBER and program latency. Since the RBER increases with the retention time, Pan et al. [32] and Liu et al. [31] propose to improve the program speed by compromising the retention time requirement. Besides the retention time, RBER also depends on the program/erase cycles and the content hold by the flash cells. It is well known that NAND flash memories gradually wear out with the increasing of program/erase cycles, and which has been exploited in [29] to improve the flash program speed. As the RBER is also dependent on the content of the flash cell, Gao et al. [28] propose to apply the fast program scheme to the flash pages with low-content-aware RBER. DiffECC is proposed in [30] to partially use the high cost write scheme during the idle time to reduce the RBER, which then could be handled by weak ECC scheme, to improve the flash read performance. AGCR [37] proposed by Li et al. tries to separate the accesses into three categories: read-only, write-only, and interleaved accesses. For read-only accesses, high-cost writes will be applied to reduce the overhead of the following read requests. While for write-only accesses, low-cost writes will be used to improve the write performance. For the interleaved accesses, the medium-cost writes will be applied to reach a comprise. Among all the above strategies, only AGCR is similar to our scheme. However, AGCR uses a uniform strong ECC scheme, which is designed for the worst cases, for all the flash pages that could have different error rates due to different write scheme applied on the pages. Therefore, AGCR is far from optimal. To separate the three different kinds of accesses, AGCR need to record the access histories for all logical pages, which will consume additional memory resources, especially with the continuous increasing capacity of SSDs. Moreover, response time of IO requests not only determined by the access latency (encoding and programming for write, sensing and decoding for read) and transfer latency, but also depends on the queueing delay, which has been identified as the key factor of the IO performance in the previous research work [42, 43]. Therefore, unlike AGCR which always conservatively apply the medium-cost write scheme for the interleaved requests, our workload-ware differentiated scheme will selectively apply the low-cost write scheme to the overlapped requests to further alleviate the queueing delay and improve the performance. Besides all the above previous research work, some research work like [33, 34] leverage other special properties of flash memory like the out-of-place updates feature to improve the flash performance.



(a) hm0 (b) hm1 (c) Financial1

Fig. 2: Read and write request distributions in the two-dimensional space (logical address and timing space).

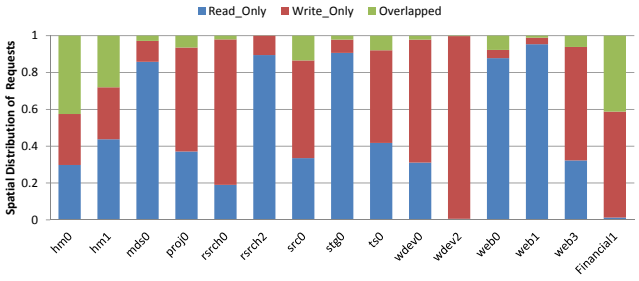


Fig. 1: Spatial distribution of the read and write requests.

III. Workload Analysis

In this section, we present the study on the access characteristics of fifteen representative workloads that are mixed with both read and write requests from the Microsoft Research (MSR) Cambridge [3] and OLTP applications [2]. Figure 1 shows the ratios of write-only, read-only, and overlapped part based on their logical page numbers (here, we assume the flash page size is 8KB). The results clearly indicate that the read and write requests could be well separated based on their logical addresses with the overlapped ratios below 10% or even 5% for most of the workloads. However, hm0, hm1, and Financial1 are three exceptions whose overlapped ratios are around 42%, 28%, and 41%, respectively. Although these three workloads have a very high overlapped ratios, the read and write requests could be easily separated in the time domain. Figure 2 shows the read and write request distributions in a two-dimensional space where x-axis and y-axis are the timing space and logical address space, respectively. In Figure 2 the red dots are the write requests, while the green dots are the read requests. From the results, we find that, for workloads with high overlapped ratio in the logical address space, the overlapping of read and write logical pages happens merely within a limited timing period, while in the other timing period, they are very well or even totally separated. From the above all results, we make the following two observations.

- Read and write requests are well separated in the logical address space for most workloads with overlapped ratios below 10% or even 5%;
- When read and write requests are highly overlapped in the logical address space, they could be easily separated in the time domain.

These two observations give us the hints to properly design the read and write separating scheme and the whole architecture of our system.

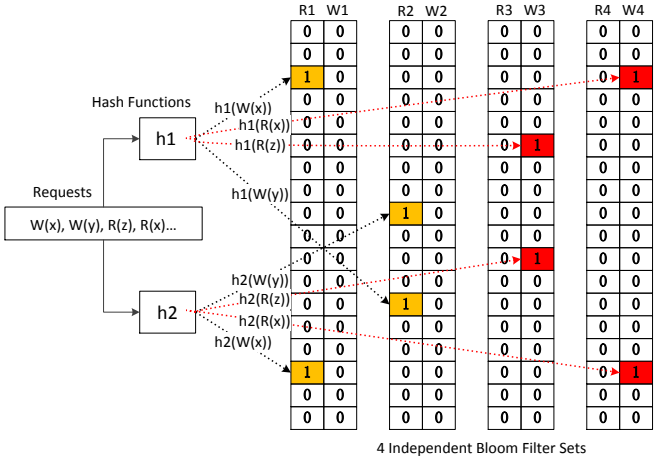


Fig. 3: Read and write separator based on multiple bloom filters.

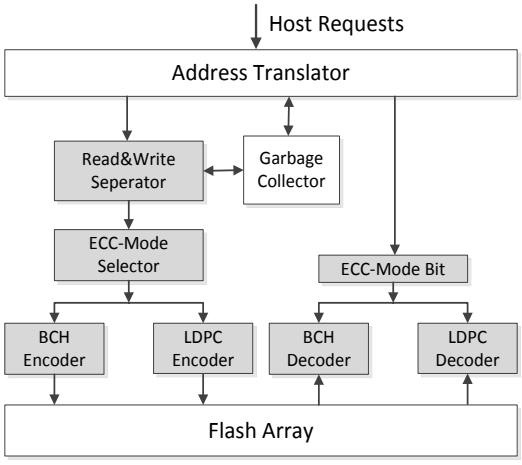


Fig. 4: Architecture of our workload-aware differentiated ECC design.

IV. Design and Implementation

In this section, we will first describe our read and write separator. Then the whole architecture and working flow of our proposed design will be presented.

A. Read and Write Separator

Multiple bloom filters have been utilized to perform hot data identification for flash-based storage system in [38] that captures both the frequency and recency of the requested data. A multiple bloom filters system consists of V independent bloom filters and K independent hash functions. Each bloom filter (BF) is a M bits array whose initial values are 0. K independent hash functions map the input elements to the corresponding K bit positions of the M bits BFs. To identify the hot data, the logical addresses of the write requests are used as the input for the K hash functions and will be mapped to K corresponding bit positions of the BFs. Each time, one BF will be selected from the V BFs in a round-robin manner, then the values of these K bit positions of the selected BF will be set to 1. To capture both the frequency and recency of the requests, V different weights will be assigned to the V BFs, where the highest and lowest weights are assigned to the latest and oldest updated BFs, respectively. What's more, a decay process is triggered to reset the oldest BF periodically.

Unlike the original multiple bloom filters architecture that only considers write requests, our read and write separator replaces all the independent bloom filters with read and write bloom filter pairs to accommodate both read and write requests. The read and write bloom filters within bloom filter pairs will only be updated by read requests and write requests, respectively. Figure 3 presents an example to show the architecture of our read and write separator and how it works. The read and write separator consists of two hash functions and four independent bloom filter pairs. Initially, all the bloom filter pairs are filled with 0. Then, a write request on logical page x comes. Logical page x will be mapped to two bit positions in the bloom filters and the write bloom filter in the first bloom filter pair will be updated. The second request is a write request on logical page y. Similarly, the write bloom filter of the second bloom filter pair will be updated. The following request is a read request on logical page z, the read bloom filter of the third bloom filter pair will be updated. The forth is also a read request and the read bloom filter of the forth bloom filter pair is updated. In this way, the read hotness of a logical page could be calculated based on the read bloom filters, while the write hotness is based on the write bloom filters. By comparing the read hotness with the write hotness, the read requests could be separated with write requests as follows:

- Read only: read_hotness $>$ 0 and write_hotness = 0
- Write only: read_hotness = 0 and write_hotness $>$ 0
- Overlapped: read_hotness $>$ 0 and write_hotness $>$ 0

According to the experimental results in [38], we configure our read and write separator with 2 hash functions, and 4 read and write bloom filter pairs with 2048 bits per bloom filter. The decay period is set as 512 requests. The recency weights are set as 2, 1.5, 1, or 0.5 based on the recency of the bloom filter pairs.

B. Architecture and Working Flow

Figure 4 gives the overview of our workload-aware differentiated ECC design. In addition to the typical components of SSDs like address translator and garbage collector, a read and write separator, ECC-model selector, ECC mode bit, and a hybrid ECC system that supports both BCH and LDPC are added in our design. A 1-bit ECC mode tag is attached to each page as metadata to help the system choose the corresponding decoder for a read request.

Whenever a host write request arrives, the write bloom filter will be updated. For write requests from both the host side and background operations like garbage collection, the ECC-Mode selector chooses the proper ECC encoder and write scheme based on the following regulations:

- For a write-only page, a low-cost write scheme with LDPC encoder will be applied;
- For an overlapped page, if the write_hotness - read_hotness $>$ m, the low-cost write scheme with LDPC encoder will be applied. Otherwise, the normal write scheme with BCH encoder will be applied;

While for a host read request, the read bloom filter will be updated. Then the corresponding decoder will be used to decode the data according to the ECC mode bit. Besides, if the read hotness of a LDPC-encoded logical page is larger than its write hotness (if its write hotness is larger than the read hotness, then the data has a high probability to be updated by the user requests, which makes the rewrite operation unnecessary.) and the difference between the read hotness and write hotness exceeds a predefined threshold n, then a rewrite operation with normal-cost write and BCH encoder will be issued to reduce the overhead of the upcoming read requests. The selections of the values for m and n will affect the performance gain of our design. A larger value of m will limit the possibility to apply our low-cost write mode. While, a small value of m may hurt the read performance. For threshold n, a larger value may hurt the read performance and a small value may introduce unnecessary rewrite operations and more garbage collection processes. In the next section, we will discuss the selection of the proper m and n.

C. Overhead Analysis

The overhead of our workload-aware differentiated ECC scheme falls into three categories: memory, firmware, and hardware. The memory overhead comes from two parts: multiple bloom filters based read and write separator, and ECC-Mode bit. The multiple bloom filters used in our implementation consist of 4 read and write bloom filter pairs with 2048 bits per bloom filter. Therefore, the memory overhead of the multiple bloom filters is merely 2KB and it is independent on the SSD capacity. Besides, each flash page needs 1 bit for the ECC-Mode bit. Assuming a 64GB flash memory with 8KB pages, the memory consumption of the ECC-Mode bits is only

1MB. The firmware overhead includes 2 hash functions, and multiple bloom filter check and update. The overhead of these simple processes is negligible. To make our scheme works, the hardware need to support both differentiated program speeds and ECC codes. This differentiated architecture has been utilized and explored in previous studies [30, 31]. What's more, the rewrites for the overlapped read requests may introduce additional write operations inside SSDs and hurt the lifetime, which will be evaluated in Section V.

TABLE I: Operation latency configuration

Operation	Latency (us)
BCH Read	45
LDPC Read Fast	75
LDPC Read Medium	110
LDPC Read Slow	145
Write Fast	650
Write Medium	1300
Write Slow	2600
Erase	3800

TABLE II: Characteristics of I/O workload traces

Workload Name	**Addr. Space**	**Request Amount**	**Read Ratio**	**Avg. Req. Size (KB)**
Financial1	16.67GB	5334857	23.16%	15.15
hm0	13.94GB	3993316	35.5%	7.99
hm1	25.44GB	609311	95.34%	15.16
mds0	33.92GB	1211034	11.89%	9.20
proj0	16.24GB	4224524	12.48%	38.04
rsrch0	16.89GB	1433655	9.32%	8.93
rsrch2	81.65GB	207587	65.69%	4.09
src0	15.63GB	1557814	11.34%	7.21
stg0	10.82GB	2030915	15.19%	11.58
ts0	21.97GB	1801734	17.58%	9.01
wdev0	16.96GB	1143261	20.08%	9.08
wdev2	33.92GB	181266	0.1%	8.15
web0	33.91GB	2029945	29.88%	14.99
web1	67.83GB	160891	54.11%	29.07
web3	169.58GB	31380	32.03%	38.14

V. Experimental Methodology and Results

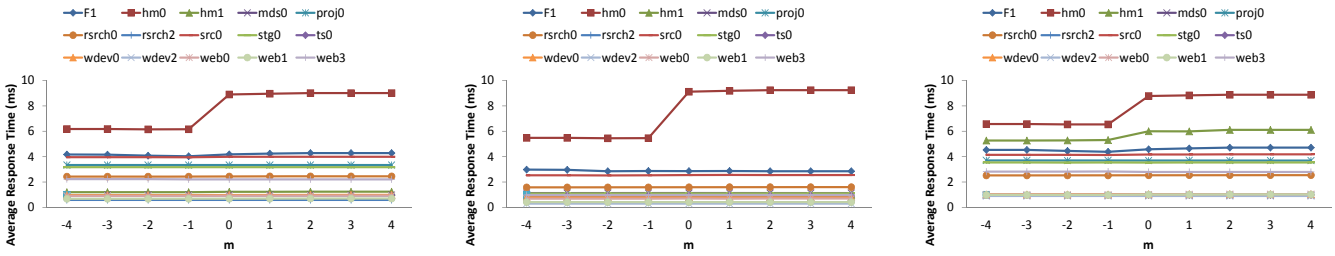
A. Experimental Methodology

We modified the Disksim with SSD extension [27] to verify our proposed design. We assume only two write schemes are supported by the device for our workload-aware differentiated ECC scheme, the normal-cost write mode and low-cost write mode that doubles the ΔV_p. Therefore, we set the normal-cost flash page write latency as 1.3 ms, low-cost write latency as 650 us, block erase latency as 3.8 ms, BCH-encoded page read latency as 45 us based on [14], and LDPC-encoded page read latency as 145 us (estimated based on [16]). For comparison, we set the latency of high-cost write, medium-cost write, and low-cost write in AGCR as 2.6 ms, 1.3 ms, and 650 us, respectively. Since the LDPC decoding latency depends the error rate of Flash pages, we estimate the average latency for the high-cost read, medium-cost read, and low-cost read for AGCR as 145 us, 110 us, and 75 us (also based on [16]), respectively. The summary of these different-cost operations is listed in Table I. The sizes of flash page and block are 8 KB and 1 MB. The SSD is configured with 8 packages. The capacity of the packages are determined by the address space of the specific workloads. The over-provision space and garbage collection threshold are set as 15% and 5%, respectively. Fifteen realistic workloads that have been analyzed in the previous section will be used in our experiments. Details of the characteristics of these workloads are depicted in Table II.

B. Experimental Results

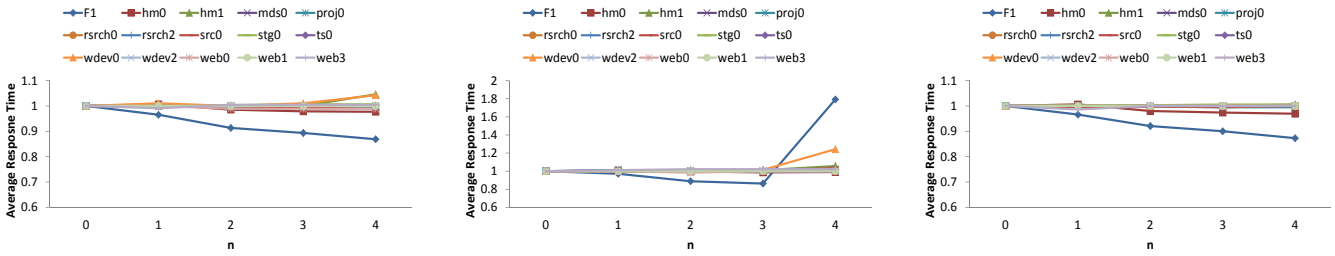
1) Parameters Exploration: In this subsection, we will show how different configurations of m and n will affect the performance of our workload-aware design. First, we fix the value of n with 2, which is the intermediate value of n, and varies the values of m from -4 to 4 based on the configuration of our multiple bloom filters. A positive value of m means that only when the write hotness of the request is larger than its read hotness, the low-cost write pattern will be applied to accelerate the write speed and reduce the queueing delay. While a negative value of m means aggressively apply the low-cost write pattern, even when the request has a higher read hotness. A large value of m may limit the chance to apply the low-cost writes for the overlapped requests, while a small value of m can lead to more high-cost reads. Figure 5(a) shows the overall average response time under different m configurations, where m changes from -4 to 4. The results indicate that the best overall performance can be achieved with m equals -1. Besides, for most of the workloads, changing the values of m only has marginal effects on the overall performance due to the small overlapped ratios of these workloads. One exception is the hm0, the average overall performance could be significantly improved when m changes from positive values to negative values. We believe there are two reasons for this exception. First is the relatively higher overlapped ratio, which means the decreasing of m can generate more low-cost write operations. The second is due to the fact that write operations are much more costly than reads. Therefore, reducing the write latency can effectively mitigate the queueing delay, which has been identified as the key factor of the IO performance in the previous research work [42, 43], and improve the overall performance. Figure 5(b) and Figure 5(c) presents the average read and write performance variations with different values of m, where the same conclusion can be made. Hence, we set the value of m as -1 as our default configuration in our following evaluations.

After setting the value of m as -1, we changes the value of n from 0 to 4. Figure 6(a) shows the normalized overall average response time with different values of n. Figure 6(b) and Figure 6(c) presents the normalized average read and write response time. The results indicate that increasing the value of n can always result in better or the same write performance. While for both the read and overall performance, only when n is less than or equal to 3, increasing the value of n can obtain better or similar results. Therefore, we set the value of n as 3 in our following experiments.



(a) Overall performance (b) Read performance (c) Write performance

Fig. 5: Performance change with various values of m, here we fix the value of n as 2.



(a) Normalized overall performance (b) Normalized read performance (c) Normalized write performance

Fig. 6: Performance change with various values of n, here we fix the value of m as -1.

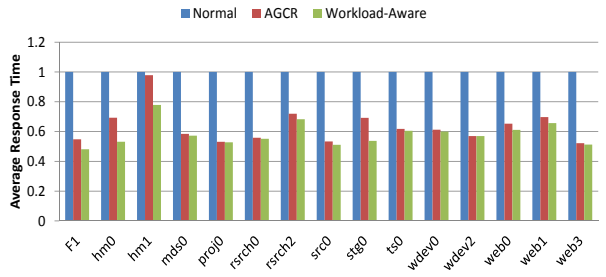
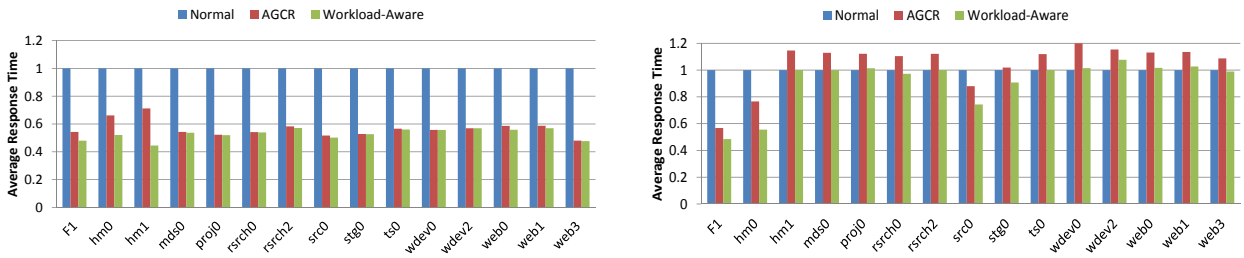


Fig. 7: Normalized overall performance comparison.

2) Performance: In this section, read, write and overall performance of our workload-aware differentiated ECC design are evaluated and compared with the normal case and the state-of-art work from Li et al. [37].

Figure 7 shows the normalized overall average response time of the normal, AGCR, and our workload-aware design. The result shows that both AGCR and our workload-aware design can significantly improve the overall performance, especially for the workloads with high write ratios like Financail1, mds0, proj0 and so on, where the improvement from AGCR and our workload-aware design could reach up to 45% and 52%, respectively. But for workloads with relatively higher overlapped ratios like Financial1, hm0, and hm1, our workload-aware design can show remarkable advantage over AGCR. For instances, our workload-aware design could improve the overall performance for Financial1, hm0, and hm1 by about 52%, 47% and 32%, respectively. While the overall performance enhancements for Financial1, hm0, and hm1 from AGCR are limited to about 45%, 30%, and 2%. There are three reasons that make our workload-aware design work better for workloads with higher overlapped ratios. The first is the adoption of the hybrid ECC architecture, which takes the advantage of both the low read latency of BCH encoded data and strong error correcting capability of LDPC code. Second, unlike the AGCR scheme that always apply the medium-cost write for the overlapped requests to make a compromise, our workload-aware scheme could aggressively apply the low-cost writes to these overlapped requests to reduce the queueing delay from these costly write operations and improve the overall performance. Finally, the introduction and proper configuration of parameter n has the potential to remove unnecessary rewrite operations and reduce the time-consuming garbage collection processes.

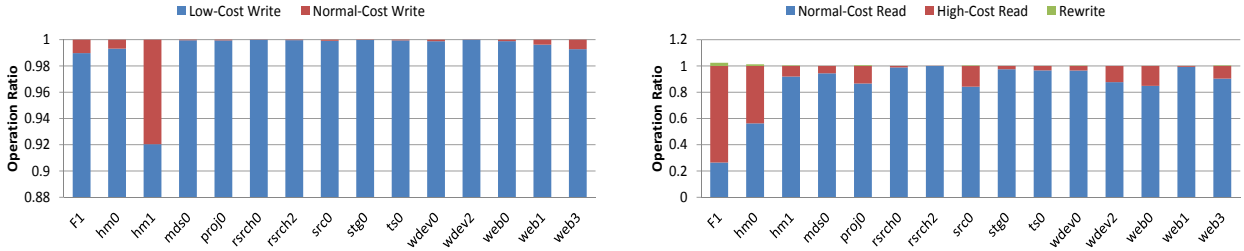
Besides the overall average performance, the normalized read and write response time are presented in Figure 8 to give more information in detail. For the write performance, both the AGCR and our workload-aware scheme can gain remarkable benefit (more than 40% reduction for most of the workloads). Similar to the overall performance, our workload-aware scheme shows remarkable advantage over AGCR for Financial1, hm0, and hm1. For the read requests, our workload-aware scheme can still maintain the same performance for majority of the workloads and could even prominently improve the performance by more than 40% for some highly overlapped workloads like Financail1 and hm0, which comes from the noticeable reduction of the queueing delay. For src0, although the overlapped ratio is about 10%, the total read ratio is also about 10%, which means that the read requests



(a) Normalized write average response time

(b) Normalized read average response time

Fig. 8: Normalized write and read performance comparison.



(a) Different cost write ratios

(b) Different cost read ratios

Fig. 9: Distributions of different cost operations.

of src0 highly overlapped with the write requests. Therefore, the read performance can be boosted with the reduction of queueing delay from the application of low-cost write pattern. While for the read performance of AGCR, except Financial1, hm0, and src0 which can benefit from the prominent reduction of queueing delay, most of the workloads shows worse read performance even compared with the normal scheme due to the costly LDPC-based read operations.

To help us better understand the performance variations of our workload-aware design, Figure 9 shows the distributions of operations of different costs, which includes normal-cost read, high-cost read, normal-cost write, low-cost write, and rewrite. For most of the workloads, more than 80% of the operations are low-cost writes and normal-cost read, which means our workload-aware scheme can properly distinguish between read-dominated and write-dominated requests and apply the differentiated read and write patterns accordingly. While for Financial1 and hm0, our workload-aware design can aggressively apply more of the low-cost write to reduce the queueing delay and achieve the best performance. For hm1, since more than 95% are read requests, therefore more normal-cost write with following more normal-cost reads is preferred by our workload-aware design. Moreover, our workload-aware design merely introduce negligible rewrite operations.

3) Impacts on Lifetime: In our workload-aware scheme, rewrite operations are issued to transfer the high-cost read operations to the normal-cost read operations, which may have some negative effects on the lifetime of flash memory. Figure 10 presents the normalized number of erase operations. Compared with AGCR, our workload-aware scheme introduces less and negligible extra erase operations. For all of the workloads, the increased erase ratios from our workload-aware design are within 2%.

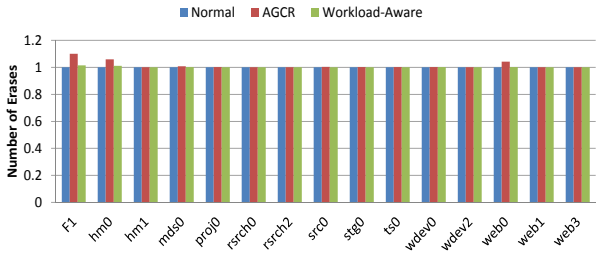


Fig. 10: Normalized number of erases.

VI. Conclusion

In this paper, we proposed a workload-aware differentiated ECC scheme. A light-weight read and write separator based on the multiple bloom filters is developed to dynamically distinguish between the write-dominated requests and the read-dominated requests. Then a low-cost write scheme with stronger ECC is applied to the write-dominated requests to improve the write performance. Our simulation results demonstrate that our workload-aware differentiated ECC design can improve the write and read performance by 48% and 11% on average, respectively, when compared with the normal design. Even compared with AGCR, our workload-aware design can still gain about 4% write performance and 11% read performance improvements.

VII. Acknowledgements

This research is sponsored in part by U.S. National Science Foundation grants CCF-1547804 and CSR-1526190. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] E. Deal, "Trends in nand flash memory error correction," *Cyclic Design, June*, 2009.

[2] "UMass Trace Repository," http://traces.cs.umass.edu.

[3] "SNIA-Block I/O Traces," http://iotta.snia.org/tracetypes/3.

[4] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, 2003.

[5] K. Eshghi and R. Micheloni, "Ssd architecture and pci express interface," in *Inside Solid State Drives (SSDs)*. Springer, 2013, pp. 19–45.

[6] K.-D. Suh, B.-H. Suh, Y.-H. Lim, J.-K. Kim, Y.-J. Choi, Y.-N. Koh, S.-S. Lee, S.-C. Suk-Chon, B.-S. Choi, J.-S. Yum *et al.*, "A 3.3 v 32 mb nand flash memory with incremental step pulse programming scheme," *Solid-State Circuits, IEEE Journal of*, vol. 30, no. 11, pp. 1149–1156, 1995.

[7] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 602–616, 2003.

[8] N. Duann, "Error correcting techniques for future nand flash memory in ssd applications," *Technology*, p. 1, 2009.

[9] F. Sun, K. Rose, and T. Zhang, "On the use of strong bch codes for improving multilevel nand flash memory storage capacity," in *IEEE Workshop on Signal Processing Systems (SiPS): Design and Implementation*, 2006.

[10] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, pp. 68–79, 1960.

[11] R. G. Gallager, "Low-density parity-check codes," *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21–28, 1962.

[12] D. J. MacKay, "Good error-correcting codes based on very sparse matrices," *Information Theory, IEEE Transactions on*, vol. 45, no. 2, pp. 399–431, 1999.

[13] J. Yang, "Novel ecc architecture enhances storage system reliability," *Proc. Flash Memory Summit*, 2012.

[14] M. Fabiano, M. Indaco, S. Di Carlo, and P. Prinetto, "Design and optimization of adaptable bch codecs for nand flash memories," *Microprocessors and Microsystems*, vol. 37, no. 4, pp. 407–419, 2013.

[15] S. Tanakamaru, Y. Yanagihara, and K. Takeuchi, "Over-10×-extended-lifetime 76%-reduced-error solid-state drives (ssds) with error-prediction ldpc architecture and error-recovery scheme," in *2012 IEEE International Solid-State Circuits Conference*, 2012.

[16] K. Zhao, W. Zhao, H. Sun, X. Zhang, N. Zheng, and T. Zhang, "Ldpc-in-ssd: making advanced error correction codes work effectively in solid state drives," in *Presented as part of the 11th USENIX Conference on File and Storage Technologies (FAST 13)*, 2013, pp. 243–256.

[17] M.-L. Chiang, P. C. Lee, R.-C. Chang *et al.*, "Using data clustering to improve cleaning performance for flash memory," *SOFTWARE-PRACTICE & EXPERIENCE*, vol. 29, no. 3, pp. 267–290, 1999.

[18] A. Ban, "Flash file system," Apr. 4 1995, uS Patent 5,404,485.

[19] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "Last: locality-aware sector translation for nand flash memory-based storage systems," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 6, pp. 36–42, 2008.

[20] Y. Zhou, F. Wu, P. Huang, X. He, C. Xie, and J. Zhou, "An efficient page-level ftl to optimize address translation in flash memory," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 12.

[21] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Performance Evaluation*, vol. 67, no. 11, pp. 1172–1186, 2010.

[22] Y.-H. Chang, J.-W. Hsieh, and T.-W. Kuo, "Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design," in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 212–217.

[23] Y. Cai, O. Mutlu, E. F. Haratsch, and K. Mai, "Program interference in mlc nand flash memory: Characterization, modeling, and mitigation," in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*. IEEE, 2013, pp. 123–130.

[24] Y. Cai, Y. Luo, E. F. Haratsch, K. Mai, and O. Mutlu, "Data retention in mlc nand flash memory: Characterization, optimization, and recovery," in *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*. IEEE, 2015, pp. 551–563.

[25] S. Li and T. Zhang, "Improving multi-level nand flash memory storage reliability using concatenated bch-tcm coding," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 18, no. 10, pp. 1412–1420, 2010.

[26] N. Xie, G. Dong, and T. Zhang, "Using lossless data compression in data storage systems: Not for saving space," *Computers, IEEE Transactions on*, vol. 60, no. 3, pp. 335–345, 2011.

[27] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance." in *USENIX Annual Technical Conference*, 2008, pp. 57–70.

[28] C. Gao, L. Shi, K. Wu, C. J. Xue, and E. H. Sha, "Exploit asymmetric error rates of cell states to improve the performance of flash memory storage systems," in *Computer Design (ICCD), 2014 32nd IEEE International Conference on*. IEEE, 2014, pp. 202–207.

[29] Y. Pan, G. Dong, and T. Zhang, "Exploiting memory device wear-out dynamics to improve nand flash memory system performance." in *FAST*, vol. 11, 2011, pp. 18–18.

[30] G. Wu, X. He, N. Xie, and T. Zhang, "Diffecc: improving ssd read performance using differentiated error correction coding schemes," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 57–66.

[31] R.-S. Liu, C.-L. Yang, and W. Wu, "Optimizing nand flash-based ssds via retention relaxation," in *Proceedings of the 10th USENIX conference on File and Storage Technologies (FASTâĂŹ12)*, 2012.

[32] Y. Pan, G. Dong, Q. Wu, and T. Zhang, "Quasi-nonvolatile ssd: Trading flash memory nonvolatility to improve storage system performance for enterprise applications," in *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*. IEEE, 2012, pp. 1–10.

[33] Q. Xia and W. Xiao, "Flash-aware high-performance and endurable cache," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on*. IEEE, 2015, pp. 47–50.

[34] ——, "High-performance and endurable cache management for flash-based read caching," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2016.

[35] P. Huang, P. Subedi, X. He, S. He, and K. Zhou, "Flexecc: partially relaxing ecc of mlc ssd for better cache performance," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 489–500.

[36] D. Kang, S. Baek, J. Choi, D. Lee, S. H. Noh, and O. Mutlu, "Amnesic cache management for non-volatile memory," in *Mass Storage Systems and Technologies (MSST), 2015 31st Symposium on*. IEEE, 2015, pp. 1–13.

[37] Q. Li, L. Shi, W. K. Xue CJason, C. Ji, Q. Zhuge, and E. Sha, "Access characteristic guided read and write cost regulation for performance improvement on flash memory," in *14th USENIX Conference on File and Storage Technologies (FASTâĂŹ16)*, 2016, p. 125.

[38] D. Park and D. H. Du, "Hot data identification for flash-based storage systems using multiple bloom filters," in *Mass Storage Systems and Technologies (MSST), 2011 IEEE 27th Symposium on*. IEEE, 2011, pp. 1–11.

[39] S. Electronics, "K9f8g08uxm flash memory datasheet."

[40] S. Swanson, "Flash memory overview."

[41] D. Ajwani, I. Malinger, U. Meyer, and S. Toledo, *Characterizing the performance of flash memory storage devices and its impact on algorithm design*. Springer, 2008.

[42] C. Gao, L. Shi, M. Zhao, C. J. Xue, K. Wu, and E. H.-M. Sha, "Exploiting parallelism in i/o scheduling for access conflict minimization in flash-based solid state drives," in *2014 30th Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2014, pp. 1–11.

[43] G. Wu and X. He, "Reducing ssd read latency via nand flash program and erase suspension." in *FAST*, vol. 12, 2012, pp. 10–10.