Improving SSD Performance Using Adaptive Restricted-Copyback Operations

Duwon Hong[1], Myungsuk Kim[1], Jisung Park[1], Myoungsoo Jung[2], and Jihong Kim[1]
[1]Department of Computer Science and Engineering, Seoul National University
[2]Department of Electrical Engineering, KAIST
Email: [1]{duwon.hong, morssola75, jspark, jihong}@davinci.snu.ac.kr, [2]m.jung@kaist.ac.kr

***Abstract*—Copyback operation can improve the performance of data migrations in SSD, but they are rarely used because of their error propagation problem. In this paper, we propose an integrated approach that maximizes the efficiency of copyback operations but does not compromise data reliability. First, we propose a novel per-block error propagation model under consecutive copyback operations. Our model significantly increases the number of successive copybacks by exploiting the aging characteristics of NAND blocks. Second, we devise a resource-efficient error management scheme that can handle successive copybacks where pages move around multiple blocks with different reliability. Experimental results show that the proposed technique can improve the IO throughput by up to 25% over the existing technique.**

***Index Terms*—Copyback, NAND flash, FTL, Storage system**

I. Introduction

Flash-based SSDs move a large amount of data internally to support various SSD management tasks such as garbage collection (GC), wear leveling and read-distrub management. Since these internal data copy operations directly interfere with I/O requests from user applications, how to efficiently handle internal data migrations is a key challenge for designing a high-performance SSD. Although there have been extensive investigations (e.g., [1]–[4]) to mitigate the impact of internal data migrations, most existing techniques do not adequately handle a new performance bottleneck of copy operations in modern SSDs. Unlike old SSDs where the copy cost was dominated by the program time t_{PROG}, in recent high-end SSDs, the data transfer time t_{DMA} between flash cells and off-chip DRAM takes a large portion of the copy cost. This shift in the performance bottleneck is due to two recent flash/SSD technology changes: 1) innovations in the flash cell design (which reduced t_{PROG}) [5] and 2) a high degree of the internal parallelism in high-end SSDs (which increases the effective data transfer time t_{DMA}).

In order to minimize t_{DMA}, copyback operations [6] are considered as one of the most effective solutions because the copyback operation can move pages without off-chip data transfers, thus eliminating t_{DMA} completely. However, copyback operations are rarely used in modern SSDs because they cause a fatal reliability problem. When pages are migrated using copyback operations, they bypass an off-chip error-correction code (ECC) module and bit errors occurred during copybacks are accumulated. If the number of the accumulated bit errors exceeds the correction capacity of the ECC module, the stored data in the copybacked page becomes *unreadable*. Furthermore, since t_{PROG} was responsible for a large portion of the data migration time in old SSDs, the performance improvement from copyback operations was marginal. However, as t_{DMA} becomes a key performance bottleneck of data migrations in modern SSDs, research on revitalizing copyback is receiving new attention. For example, FastGC [7] shows that copyback operations can be useful in reducing the GC overhead by limiting the number of consecutive copyback operations so that the accumulated bit errors do not exceed the error correction capability of a common ECC scheme.

In this paper, we propose an integrated approach that maximizes the efficiency of copyback operations but does not sacrifice data reliability. Although our approach is based on the same motivation as FastGC [7], we improve the existing technique in two major aspects. First, we propose a novel per-block error propagation model under consecutive copyback operations. Our model aggressively exploits the aging characteristics of NAND flash memory in deciding the copyback threshold of a NAND block. (We call the maximum number of consecutive copyback operations allowed for a NAND block as the copyback threshold of the NAND block.) From our characterization study with 3D TLC NAND chips [8], we observed that the copyback threshold of a NAND block cannot be accurately predicted by using a simple NAND aging model based on the number of P/E cycles. For example, even when two blocks had the same number of P/E cycles, their copyback threshold values can range from 3 to 5. By exploiting per-block differences during run time, our model significantly increases the copyback threshold of most NAND blocks over FastGC.

Second, we devise an efficient error management scheme that can handle successive copyback operations where pages move around multiple blocks with different reliability. When a page is migrated through blocks with different copyback thresholds, our scheme accurately maintains the remaining copyback balance of the page regardless of different copyback thresholds of migrated blocks. In managing the remaining copyback balance of a page, our scheme employs a per-block scheme instead of a more direct per-page scheme as used in FastGC. Unlike the common perception, the per-block management scheme, which can significantly reduce the memory and flash requirement over the per-page management scheme, improves both the performance and lifetime of SSDs. As a side effect of the per-block management, our scheme naturally separates data with different lifetimes into different blocks, thus achieving the high GC efficiency. By mitigating the flash requirement of the per-page management, our scheme improves the write amplification factor (WAF) as well. In the rest of the paper, we call the proposed copyback scheme as rCPB.

In order to evaluate the effectiveness of the rCPB scheme, we implemented an rCPB-aware FTL, rcFTL. We have evaluated rcFTL using various benchmarks on our SSD emulation environment [9]. Our experimental results show that rcFTL can improve the overall I/O throughput up to 25% over FastGC. In addition to basic extensions for supporting rCPB, rcFTL implements an intelligent data-migration mode selector for maximizing the effect of rCPB on the SSD performance. The mode selector decides whether to use off-chip copy operation or rCPB when performing a data migration depending on the I/O intensity of host I/O workloads. The experimental results show that the mode selector can improve the I/O throughput by up to 12% over when it is not used.

The rest of this paper is organized as follows. In Section II, we review the data migration in modern SSD and explain why NAND aging-aware copyback is needed. Section III describes the proposed rCPB operation. In Section IV, we present our design and implementation of rcFTL in details. Sections V and VI describe our evaluation results and related work, respectively. We conclude in Section VII with a summary.

II. Motivations

A. Data Migration in Modern SSD

A typical data migration in SSDs is performed by an off-chip data copy. An SSD firmware reads data from a source page and transfers the data to a DRAM buffer through a channel bus. Before the data are sent to the DRAM buffer, errors are corrected by the ECC module of the flash memory controller (FMC). In the program phase, the SSD firmware takes a reverse data path from the DRAM buffer to the target page. The data copy time t_{COPY} can be expressed as follows: $t_{COPY} = t_R + t_{DMA^{out}} + t_{DMA^{in}} + t_{PROG}$ where t_R, $t_{DMA^{out}}$ and $t_{DMA^{in}}$ are a data transfer time from NAND cells to a per-plane register and a DMA out/in time between the register and DRAM buffer, respectively. However, a large number of data migrations may occur at the same time in a modern SSD. A high degree of the parallelism in data migrations may significantly increase $t_{DMA^{in}}$ and $t_{DMA^{out}}$ because of contentions on the channel level as well as the serial bus to/from the DRAM buffer. This is because the bandwidth of the DRAM is limited and the efficiency of the DRAM degrades when many masters request DRAM at the same time.

On the other hand, when a copyback operation is used, a data migration can be performed without requiring neither $t_{DMA^{out}}$ nor $t_{DMA^{in}}$. The FTL can read data from the source page to the per-plane local register and directly write back to the destination page from the per-plane local register. Since the copyback operation transfers data within a given plane, even when multiple data migrations occur at the same time, all data migrations can be completed by (t_R + t_{PROG}). Thus, it can significantly reduce the overhead of data migrations especially for modern SSD of multiple channels and multiple ways.

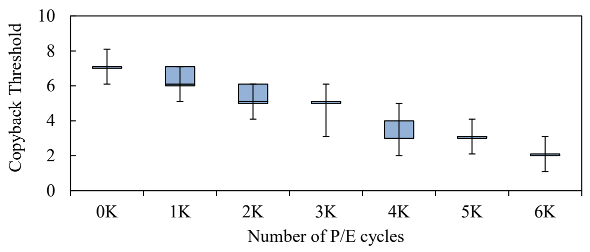


Fig. 1: Copyback threshold variations on different P/E cycles.

B. Need for Block Aging-Aware Copyback

Generally, the number of P/E cycles has been mainly used as the indicator of NAND aging. During NAND operations, the high voltage used in the erase operation damages the tunnel oxide of the NAND cells, thus increasing the Bit Error Rate (BER) observed in subsequent reads. As the number of P/E cycles increases, the tunnel oxide layer eventually reaches a state in which the cells can no longer store information reliably. Since erase operations are responsible for the wear of NAND cells, the number of P/E cycles has been regarded as a good proxy indicating the wear of NAND cells.

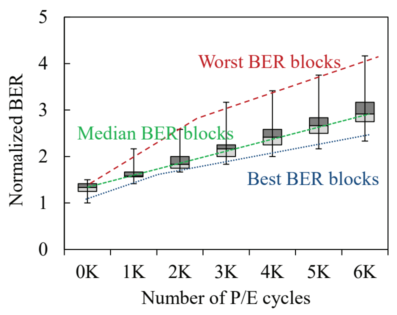
However, the number of P/E cycles alone cannot accurately represent the exact wear status of NAND cells. For example, when two NAND blocks experience the same number of P/E cycles, their BER could be significantly different [10]. This difference is mainly caused by process variations in the manufacturing and is accelerated by various user environments such as operating temperature. From our characterization study using 3D TLC NAND chips [8], we observed that the copyback threshold of a block cannot be accurately estimated by only using the P/E cycles as a wear indicator of NAND cells. Fig. 1 shows that, even at the same P/E cycles, there is a large variation on the copyback threshold count. In FastGC, since a single copyback threshold value was used for all the blocks with the same P/E cycles, the copyback threshold was conservatively selected, thus missing many opportunities for additional copybacks on most blocks.

The rCPB scheme proposed in this paper was mainly motivated from how to exploit these missed copybacks. From our characterization study, which will be described in Section III, we observed that the copyback threshold of a NAND block can be accurately predicted when the P/E cycles of the NAND block is augmented with the BER value measured right after a program operation. Using this extended NAND wear indicator, most of the missed copybacks in FastGC can be successfully utilized under the rCPB scheme.

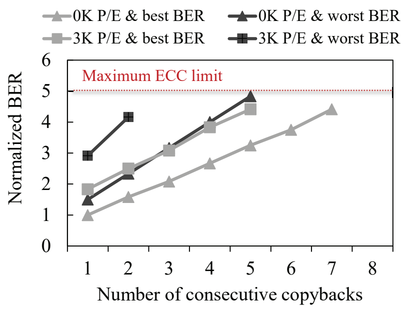
III. rCPB: Copyback with a Limit

A. Error-Propagation Characteristics

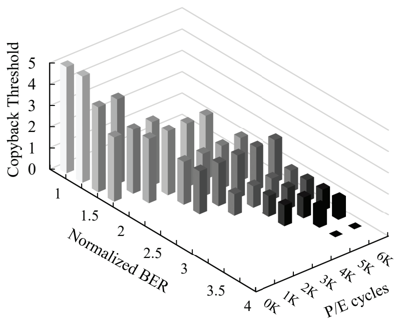
In order to manage the flash reliability problem caused by successive copyback operations, it is important to understand the NAND error propagation characteristics when the same page experiences consecutive copybacks without error correction by the ECC module. We conducted a NAND reliability characterization study using 30 actual 3D TLC NAND chips [8] to better understand the error propagation



(a) BER variations over P/E cycles.



(b) BER variations over block characteristics.



(c) Copyback thresholds distribution.

Fig. 2: Key results of the NAND characterization study.

characteristics under successive copybacks. In order to take into account of block-to-block variations as well as page-to-page differences, we selected 128 blocks from each chip where their physical locations were evenly distributed within the chip. For a selected block, we tested all the pages in the block. In our study, a total of 3,840 blocks and 2,211,840 pages were evaluated to obtain statistically significant experimental results.

In order to derive the proposed rCPB model, we first evaluated the reliability differences within NAND blocks with the same P/E cycles. As a measurement of the block reliability, we used the BER value of a block immediately after a program operation. Since the BER value is computed right after the block is programmed, no BER degradation is added from the retention errors or read disturb errors, so that the measured BER represents the block reliability level more accurately. Fig. 2(a) shows how BERs fluctuate within blocks with the same P/E cycles. As expected, large BER variations exist among the same aged blocks by P/E cycles. For example, the BER of the worst block is 1.8-times larger than that of the best block when the number of P/E cycles is 6K. As shown in the box plot, BER values of most blocks are clustered around the average BER of a given P/E cycles. On the other hand, the worst BER values make BER distribution long-tailed ones. This contributes many missed copybacks in FastGC.

In order to characterize the effect of the block reliability level on the copyback threshold, we observed how BERs change over successive copybacks when the block reliability level changes. We divided the blocks according to the measured BER characteristics and evaluated the difference of error accumulation characteristics by copyback operations for each group. Fig. 2(b) illustrates our evaluation results on the best BER blocks and the worst BER blocks under the initial (i.e., 0K) P/E cycle condition and 3K P/E cycle condition, respectively. (We used the 3-month retention requirement in this evaluation.) Under all conditions, the worse BER characteristic of the blocks, the larger the error accumulation due to the copyback operations. Considering the error accumulation level due to copyback operations for each group and the correction capability of the ECC module, it is possible to make the copyback thresholds for each condition.

Fig. 2(c) shows how the copyback threshold value changes under each P/E cycle condition when considering the retention requirement with block BER characteristic. The retention requirement was assumed to be one year at 30°C. Even at the same P/E cycles, the copyback threshold varies greatly depending on BER characteristic of the NAND blocks. When P/E cycle is 3K, the best BER block can use two more copyback operations than the worst BER block. Therefore, by distinguishing the block reliability level, the total number of copybacks can increased over the block-unaware worst-case setting of FastGC. In order to identify the copyback threshold accurately, we considered the properties of P/E cycles, retention time requirement, all possible data migration cases between source and destination pages, and the different characteristics between NAND blocks.

B. RCPB Operation Model

From our characterization study on the copyback error propagation, we constructed the copyback threshold table, $CTT(x, e, t)$, which indicates the maximum number of consecutive copyback operations that does not cause any reliability problem for x P/E-cycled blocks of BER value e under the condition of t-month retention requirement. Table I summarizes our proposed rCPB operation model with the different retention requirements. If 1-year retention is required at 2K P/E cycles, the copyback threshold of NAND block is determined from 2 to 4 based on the value of p. If the data migration is required more than the copyback threshold, the page must be migrated using an off-chip data copy, thus the accumulated bit errors can be corrected by the ECC module. As the table shows copyback threshold can be increased on 3-month retention. In this paper, we used 1-year retention as a basic requirement in accordance with JEDEC standards.

TABLE I: The proposed rCPB operation model.

Retention requirement	Block BER characteristic	P/E cycles					
		[0K∼0.4K]	(0.4K∼1K]	(1K∼2K]	(2K∼3K]	(3K∼4K]	(4K∼5K]
1 year	Best block	5	4	4	3	3	2
	Median block	5	4	3	3	2	1
	Worst block	3	2	2	1	1	0
3 months	Best block	6	5	4	4	3	2
	Median block	6	5	4	3	3	2
	Worst block	4	3	2	2	1	1

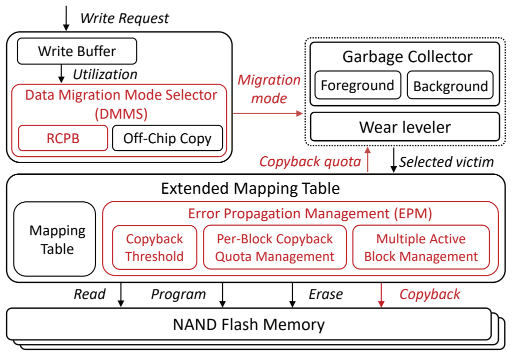


Fig. 3: An organizational overview of rcFTL.

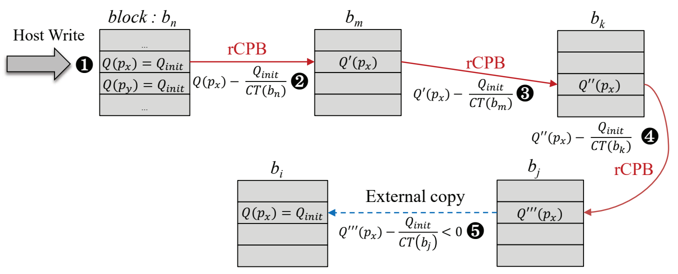

block : b_n
Host Write
❶
$Q(p_x) = Q_{init}$
$Q(p_y) = Q_{init}$
rCPB
$Q(p_x) - \frac{Q_{init}}{CT(b_n)}$ ❷
b_m
$Q'(p_x)$
rCPB
$Q'(p_x) - \frac{Q_{init}}{CT(b_m)}$ ❸
b_k
$Q''(p_x)$
$Q''(p_x) - \frac{Q_{init}}{CT(b_k)}$ ❹
rCPB
b_i
$Q(p_x) = Q_{init}$
External copy
$Q'''(p_x) - \frac{Q_{init}}{CT(b_j)} < 0$ ❺
b_j
$Q'''(p_x)$

Fig. 4: An illustrative example of the quota transition.

IV. Design and Implementation of rcFTL

Based on the proposed rCPB model presented in Section III, we implemented an rCPB-enabled flash translation layer (FTL), called rcFTL, which is based on the existing page-level mapping FTL. Fig. 3 shows an overall organization of rcFTL. RcFTL consists of two additional modules, the error propagation management (EPM), and the data migration mode selector (DMMS). The EPM module is in charge of checking the rCPB availability for a data migration while the DMMS module selects the most appropriate data migration mode for a given data copy request.

A. EPM module

1) Quota-Based Determination of RCPB Availability: The EPM module plays a key role to ensure the data reliability while rcFTL maximally uses rCPB operations. When an rCPB operation is desired in moving a page, the EPM module checks its availability. A key challenge in determining the rCPB availability is that pages are moved across various blocks that have different copyback thresholds. If a page migration is (somehow) managed to move pages only within NAND blocks with the same threshold, determining the rCPB availability is straightforward. All we need is to keep every page's rCPB count less than the threshold of those blocks. However, such a page migration management is rather impractical because it significantly obstructs a flexible page allocation of an FTL.

In order to effectively determine the rCPB availability of page migrations across blocks with different thresholds, the EPM module employs *quota-based rCPB model* which regards the copyback threshold of a block as the quota spent upon the rCPB from the block. For example, if the copyback threshold of a block is CT, the EPM module considers that a copyback operation from the block deducts $\frac{1}{CT}$ of the maximum quota which is initially given the same amount for every block. This is based on our observation that the error-propagation in successive rCPB operations at the same block is almost linear (even though the error increase per rCPB operation varies in different blocks).

Fig. 4 shows how rcFTL deals with the page migrations with the quota-based rCPB model. The EPM module keeps track of the copyback quota $Q(p_i)$ for every page where p_i indicates a page whose index is i in an SSD. When a page p_x is programmed by a host write, $Q(p_x)$ is initialized with Q_{init} (❶ in Fig. 4). Once rcFTL moves p_x using an rCPB operation from block b_n to b_m, the EPM module decreases $Q(p_x)$ by deducted quota, $\frac{Q_{init}}{CT(b_n)}$ (❷), where $CT(b_n)$ is the copyback threshold of b_n. We can obtain $CT(b_n)$ just by retrieving the predefined CTT with $PE(b_n)$ and $BER(b_n)$[1]. For a simple management of $Q(p_i)$, we set Q_{init} to the least-common-multiple value of all the present values in the CTT. As long as $Q(p_x) \geq 0$, p_x can be moved through successive rCPB operations (❸ and ❹). On the other hand, if the deducted quota of the source block b_j, $\frac{Q_{init}}{CT(b_j)}$, is large enough to make $Q(p_x)$ less than 0, the EPM module only allows off-chip copy so that $Q(p_x)$ is initialized with Q_{init} (❺).

2) Per-block Quota Management: Although the quota-based approach is effective in determining the rCPB availability, managing $Q(p_i)$'s in a per-page fashion may introduce non-trivial space overhead, considering the capacity increase of the modern SSDs. For example, suppose that Q_{init} is 30 and rcFTL manages 4-KB logical-to-physical mappings in a 16-TB SSD. In such a case, at least more than 2.5-GB[2] memory space is additionally required for the per-page quota management.

In order to avoid the overhead of per-page quota management, EPM module employs a *per-block* quota management approach. That is, the amount of copyback quota deducted by rCPB operations is managed at the block level, not at the page level. Since the number of entry for the per-block management is at least two orders of magnitude smaller than that for the per-page management, the per-block management technique significantly reduces the memory footprint for the copyback quota and minimizes the computing overhead of bookkeeping operations to a negligible level. Since all the pages in a block are assumed to have the same amount of the copyback quota in the per-block quota management, when a source page p in a victim block $b_v[Q(b_v), dQ(b_v)]$ with the copyback quota $Q(b_v)$ and the deducted quota $dQ(b_v)$ is migrated by rCPB, the page p should be moved to a page in a block b_d where $Q(b_d) = Q(b_v) - dQ(b_v)$. In order to efficiently support this additional constraint, the EPM module manages *multiple active blocks* per plane at the same time. Fig. 5 shows an example of how data migrations are performed using rCPB operations in the per-block quota management. If the Q_{init} is given by 6, the EPM module maintains five active blocks whose copyback quota is divided into 0, 2, 3, 4, and 6. When

[1]P/E cycles of block b_n, $PE(b_n)$, is maintained in typical SSDs, so rcFTL needs to additionally keep track of $BER(b_n)$ as explained in Section III.

[2]$(5[\frac{bit}{page}] \times 16 \times 10^{12}[\frac{byte}{SSD}] \times (4 \times 10^9[\frac{byte}{page}])^{-1} = 2.5 \times 10^9[\frac{byte}{SSD}])$

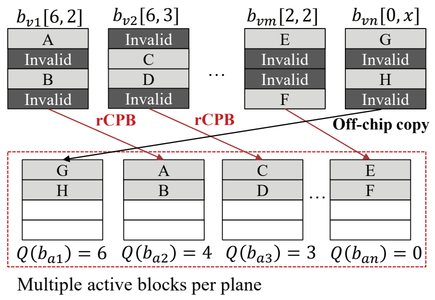


Fig. 5: Data migrations in the per-block management.

a block $b_{v2}[6,3]$ is selected as a GC victim block, its valid pages, C and D, are moved to the active block b_{a3} which has the quota of 3 by rCPB operations. On the other hand, if the block $b_{vn}[0,x]$ is selected as a GC victim block, its valid pages are moved using off-chip copies to the active block b_{a1} which has the initial quota, 6.

B. Data Migration Mode Selection

In order to take full advantages of rCPB, the DMMS module intelligently chooses when to use rCPB over a normal off-chip copy depending on the write buffer utilization ratio u. When u is low, which indicates that the current host I/O workload is not intensive, the DMMS module selects the off-chip copy mode so that more future data migrations can be supported by rCPB. On the other hand, when u is high, the DMMS module chooses the rCPB mode for higher performance. In our current implementation, the utilization threshold ratio for the mode selection was set to 50%. (That is, if u is higher than 50%, the rCPB mode is used for data migrations.) Since rcFTL employs the per-block quota management scheme and most data migration decisions are made in a block granularity, the DMMS module makes its mode selection decisions in a per-block level as well. When a data migration decision is made (e.g., by a foreground GC task), the DMMS module selects a proper mode based on the current u value. In order to filter out abrupt noise-like changes in u, the DMMS module makes its mode selection based on a t-second moving average of u. In the current implementation, t is set to an average block write time.

In rcFTL, both the GC and wear leveler operate in an rCPB-aware fashion. For urgent management tasks (such as a foreground GC task), the rCPB mode is actively used regardless of the current u ratio value. On the other hand, when background management tasks (such as a background GC task) are invoked, the DMMS module decides proper modes as explained above.

V. Experimental Results

A. Experimental Setup

In order to evaluate the effectiveness of the proposed rcFTL technique, we implemented rcFTL as a host-level FTL on a custom flash storage system [9]. For our evaluation, we configured our flash storage system to support a 128-GB storage capacity only for efficient experimental evaluations. Our emulated storage system was configured to have eight

TABLE II: I/O characteristics of traces used for evaluations.

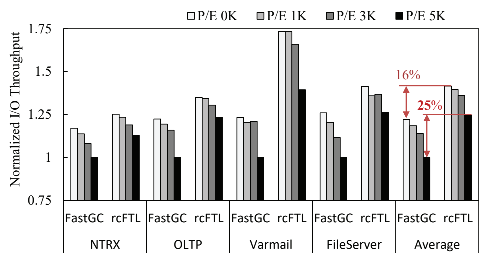
	OLTP	NTRX	Varmail	Fileserver
Read:Write	7:3	0.5:9.5	4:6	4:6
WAF	2.0	2.3	2.7	5.4

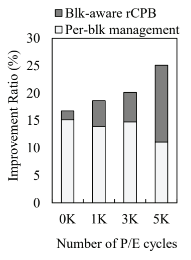
channels with eight NAND flash chips per channel. Each NAND flash chip has 1024 blocks which are composed of 128 16-KB pages and NAND interface which supports up to 533 MT/s. The average t_{PROG} was set to 660 us [8] and the size of the write buffer was set to 10 MB. We evaluated rcFTL using four I/O traces generated from Sysbench and Filebench. As shown in Table II, each workload has different ratios between read and write and different WAF values. Using these workloads, we evaluated the overall I/O throughput for six different P/E cycle conditions where the copyback threshold counts are distinguished and compared them with the existing techniques. All measurements were normalized over a page-level mapping FTL which always migrates data using the off-chip copy.

B. Evaluation Results

Fig. 6(a) shows the normalized I/O performance for each workload under various P/E cycle conditions. The proposed rcFTL has better performance compared to the existing FastGC method for all workloads because it uses block-aware copy-back threshold effectively and the per-block management scheme for the copyback quota improves the WAF value. The overall I/O throughput was improved by 43% on average of four workloads in initial P/E cycle over the baseline FTL. It also improved I/O throughput by 25% over the FastGC method when P/E cycles is 5K. When blocks are young, most data migrations can be supported by copybacks even in the worst blocks. As the copyback threshold increases, the degree of performance improvement due to rCPB degrades, most of the improvements in young blocks over FastGC comes from per-block management of rcFTL. On the other hand, as shown in Fig. 6(b), as blocks get older, the impact of block-aware rCPB scheme grows. For example, the performance improvement by block-aware rCPB is 14% when the P/E cycle is 5K.

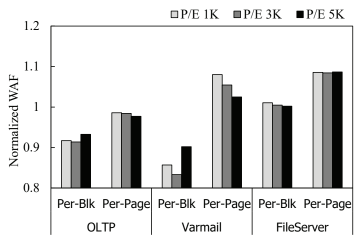
In order to analyze the effect of per-block management of rcFTL, we compared the WAF value of rcFTL and existing per-page management. Fig. 7(a) shows normalized WAF value based on baseline FTL. Overall, per-block management scheme of rcFTL showed lower WAF than per-page management for all conditions. Although the overhead of per-page management was amplified due to the small number of pages per block by limited capacity, the WAF value was increased in per-page management as the copyback threshold increases in OLTP and Varmail. On the other hand, the per-block management of rcFTL decreased WAF value as the copyback threshold increases. This is the result of separating data with different lifetimes into different blocks through multiple active block management of per-block scheme without consuming flash resources. However, there was no WAF reduction effect of per-block management on FileServer. This is because the workload has a strong random update characteristic that does not have any significant locality.

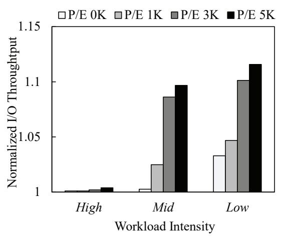




(a) Normalized I/O throughput. (b) Improvement breakdown.

Fig. 6: Performance comparison between FTLs.





(a) Per-block management. (b) Migration mode selector.

Fig. 7: Effectiveness of each FTL module.

In order to understand how the mode selector proposed in rcFTL performs, we compared the performance of rcFTL with rcFTL− (which uses rCPB in a greedy fashion). Fig. 7(b) shows normalized performance gain of rcFTL over rcFTL− under varying I/O intensity. In order to generate workload fluctuations, which are needed to properly evaluate the DMMS module, we generated three synthetic workloads, *High*, *Mid* and *Low*, using Fio benchmark. In *High*, 70% of I/O requests were issued without inter-request idle times while 30% were issued with some idle times. For *Mid* and *Low*, the ratio between two requests is 50:50 and 30:70, respectively. When the I/O intensity is lower than the threshold, since the off-chip copy mode is more likely to be used in rcFTL, rCPB-eligible blocks tend to increase over rcFTL− because the copyback quota of more blocks are reset. The increased number of rCPB-eligible blocks, in turn, improves the I/O throughput when the I/O intensity is higher than the threshold. Fig. 7(b) shows that the performance gain is higher when the workload intensity is *Low*. The performance is further improved, especially in small copyback threshold conditions, which shows that the mode selector works effectively.

VI. Related Work

There have been several studies to improve the performance of flash-based storage systems with the copyback operation. However, many existing techniques [11]–[13] are not applicable for modern NAND flash memory because they assumed an ideal SLC NAND flash memory where no error propagation occurs from successive copyback operations. Other studies such as Jang *et al.* [14] considered the error propagation problem in their techniques. However, their solutions was to bring data out to the ECC module to check the validity of data, thus minimizing the potential benefit of using copyback. In recent study of Wu *et al.* [7], they proposed a technique that can use copyback without error propagation based on NAND characterization for the first time. However, there is a lot of room for improvement because their method is a naive approach and there is overhead for error propagation management. Our technique differs from the existing technique in that it maximizes the potential benefits of copyback by taking both block characteristics and host workload characteristics into account, and has full control over error propagation issues with minimal overhead.

VII. Conclusions

We have presented rCPB to minimize the performance degradation from data migrations in modern SSDs. From a NAND characterization study, we developed an rCPB operation model that takes as the key inputs block characteristics and data retention requirement. Based on the rCPB operation model, we have implemented an rCPB-aware FTL, rcFTL, which intelligently manages when to use rCPB for a given I/O workload requirement. Our experimental results show that rcFTL can improve the overall I/O throughput up to 25% over the existing technique.

VIII. Acknowledgements

This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1701-11.

References

[1] A. Gupta *et al.*, “Dftl: A flash translation layer employing demand-based selective caching of page-level address mappings,” in *Proc. Int’l Conf. Architectural Support for Programming Languages and Operating Systems*, 2009.

[2] J.-U. Kang *et al.*, “A superblock-based flash translation layer for nand flash memory,” in *Proc. Int’l Conf. Embedded Software*, 2006.

[3] J. Kim *et al.*, “A space-efficient flash translation layer for compactflash systems,” *IEEE Trans. Consumer Electronics*, vol. 48, no. 2, pp. 366–375, 2002.

[4] S.-W. Lee *et al.*, “Fast: An efficient flash translation layer for flash memory,” in *Proc. Int’l Conf. Embedded and Ubiquitous Computing*, 2006.

[5] H. Kim *et al.*, “Evolution of nand flash memory: from 2d to 3d as a storage market leader,” in *Proc. Int’l Memory Workshop*, 2017.

[6] Tn-29-15: Nand flash internal data move idm overview. [Online]. Available: https://www.micron.com/∼/media/documents/products/technical-note/nand-flash/tn2915.pdf

[7] F. Wu *et al.*, “Fastgc: Accelerate garbage collection via an efficient copyback-based data migration in ssds,” in *Proc. Design Automation Conf.*, 2018.

[8] D. Kang *et al.*, “256gb 3b/cell v-nand flash memory with 48 stacked wl layers,” in *Proc. Int’l Solid-State Circuits Conf.*, 2016.

[9] S.-W. Jun *et al.*, “Bluedbm: An appliance for big data analytics,” in *Proc. Int’l Symp. Computer Architecture*, 2015.

[10] Y. Pan *et al.*, “Error rate-based wear-leveling for nand flash memory at highly scaled technology nodes,” *IEEE Trans. on Very Large Scale Integration Systems*, vol. 21, no. 7, pp. 1350–1354, 2013.

[11] Y. J. Seong *et al.*, “Hydra: A block-mapped parallel flash memory solid-state disk architecture,” *IEEE Trans. Computers*, vol. 59, no. 7, pp. 905–921, 2010.

[12] A. R. Abdurrab *et al.*, “Dloop: A flash translation layer exploiting plane-level parallelism,” in *Proc. Int’l Symp. Parallel and Distributed Processing*, 2013.

[13] W. Wang and T. Xie, “Pcftl: A plane-centric flash translation layer utilizing copy-back operations,” *IEEE Trans. Parallel and Distributed Systems*, vol. 26, no. 12, pp. 3420–3432, 2015.

[14] W. T. Chang *et al.*, “An efficient copy-back operation scheme using dedicated flash memory controller in solid-state disks,” *Int’l Journal of Electrical Energy*, vol. 2, no. 1, pp. 13–17, 2014.