ShadowGC: Cooperative Garbage Collection with Multi-level Buffer for Performance Improvement in NAND flash-based SSDs

Jinhua Cui*, Youtao Zhang†, Jianhang Huang*, Weiguo Wu*, Jun Yang†
*Xi'an Jiaotong University
cjhnicole@gmail.com, wgwu@xjtu.edu.cn, huangjhsx@gmail.com
†University of Pittsburgh
zhangyt@cs.pitt.edu, juy9@pitt.edu

***Abstract*—Garbage collection, an essential background activity in NAND flash based SSDs, often introduces large runtime overhead. Recent studies showed that it is beneficial to separate the flash pages that have dirty copies in the write buffers from those that do not. However, the existing schemes exploring this observation have limitations, which prevent them from maximizing the performance improvement.**

In this paper, we address the above challenge through ShadowGC, a novel GC design that exploits the pages in both host-side and device-side write buffers and adopts different read and write strategies to minimize the GC overhead. When garbage collecting flash pages that have dirty copies in the device-side write buffer, ShadowGC reads data from the write buffer. When garbage collecting flash pages that have dirty copies in the host-side write buffer, ShadowGC moves them to dedicated blocks and speeds up the movement with fast-write operations. Our experimental results show that, on average, ShadowGC reduces the write amplification by 16.2% and the GC latency by 20.5% over the state-of-the-art.

1. Introduction

NAND flash-based solid-state drives (SSDs) are increasingly adopted in modern computer systems, ranging from personal computers to large-scale cloud servers. Comparing to traditional hard disk drives (HDDs), SSDs have many advantages such as lower power consumption, smaller shock resistance and noise, and 5000x more IOPS at 1% of the latency [1]. However, NAND flash memory has several shortcomings due to its physical characteristics: write-once property (a previously written page cannot be updated until the block with that page is erased), and asymmetric operation units (space is reclaimed in units of multi-page erase blocks, where a page is the access unit). These characteristics create the requirement for an Flash Translation Layer (FTL), which is responsible for mapping logical addresses from a file system to physical addresses in SSDs.

Garbage collection (GC) is an essential background activity in FTL [2]. When the number of clean flash blocks falls below a threshold, GC is invoked to reclaim used blocks. The overhead of GC comes from two folds: (1) a block can be erased only after copying the valid pages in the block to other places. Moving these pages leads to extra flash write operations, which is referred to as write amplification issue. (2) An I/O operation, in particular, a read operation, if having resource conflicts with ongoing GC operations, may suffer from a long access latency, which leads to large I/O access latency variations.

Over the past decade, much effort has been invested to minimize the GC overhead. This includes reducing the number of GC operations through better page layout [4], cell reuse [5], [6], and tail delay minimization [7], [8], [9]. Recent studies [10], [11] proposed to exploit the valid pages that have dirty copies in the write buffers. Since the pages in the write buffer tend to be written back soon, it is beneficial to separate their corresponding copies in the SSD from other valid pages. However, the existing schemes that explore this observation have limitations. The skip policy adopted in [11] is applicable only to systems with non-volatile write buffers. The delay policy adopted in [10] was designed for systems with super large write buffers, which becomes less effective for mainstream commodity systems. Both schemes did not differentiate the write buffers that exist at host side and at device side in the flash-based storage system.

In this paper, the flash pages that have dirty copies in the write buffers are referred to as *shadows*. We propose ShadowGC, a novel GC design to effectively exploit the shadows for performance improvement. We summarize our contributions as follows.

- When garbage collecting the shadow pages that have dirty copies in the host-side write buffer, we relocate the shadows to dedicated blocks and choose fast write mode to program them, which effectively mitigate both write amplification and latency issues. Given shadows have short lifetime, most pages in the dedicated blocks would become invalid when they are to be reclaimed. By adopting fast write mode, we significantly speed up page write operations and thus reduce the average GC latency.
- When garbage collecting the shadow pages that have dirty copies in the device-side write buffer, we choose to read the contents from the write buffer, which mitigates both write amplification and latency issues too. By merging the write-back operations of the buffer with the page relocation operations in GC, we effectively reduce the

978-3-9819263-0-9/DATE18/©2018 EDAA

number of extra writes. By copying the data from the write buffer, we skip the long latency flash read and, in particular, the ECC check and correction overhead, which reduces the average GC latency.

- We evaluate the proposed ShadowGC scheme and compare it to the state-of-the-art. Our experimental results show that, on average, ShadowGC reduces the write amplification by 16.2% and the GC latency by 20.5% over the state-of-the-art.

The rest of this paper is organized as follows. Section 2 presents the background and motivation. Section 3 elaborates the proposed ShadowGC scheme for flash storage devices. Experiments and result analysis are presented in Section 4. Section 5 concludes the paper.

2. Background and Motivation

In this section, we first introduce the background activities in the FTL of SSD. We then study the real-world workload characteristics that motivate the ShadowGC design.

2.1. Background Activities in SSDs

In addition to the host generated I/O activities, NAND flash-based SSDs should handle the FTL generated background activities, which causes write amplification and latency issues. We go over two important modules as below.

Garbage collection (GC): GC is needed to reclaim wasted space occupied by invalid pages, which are inevitably created after logical-to-physical remapping. When the number of clean flash blocks falls below a certain threshold, GC is invoked to generate clean blocks by selecting a block with invalid pages and erasing it after copying valid pages to a free block. As a result, host generated I/O requests which access GC-ing chips will be blocked. A lot of works have shown that GC can induce significant performance variation problem and the worst-case latency can be much higher than HDDs, which is unbearable for high performance computing and enterprise environments [9]. In addition to the GC latency, write amplification is another important problem to SSD lifetime.

Refresh: The goal of refresh is to improve flash lifetime by periodically reading, correcting, and in-place reprogramming or remapping the stored data before the retention induced raw bit error rates (RBER) exceed the Error Correction Code (ECC) capability [3]. Refresh also induce significant performance variation and write amplification.

2.2. Motivation

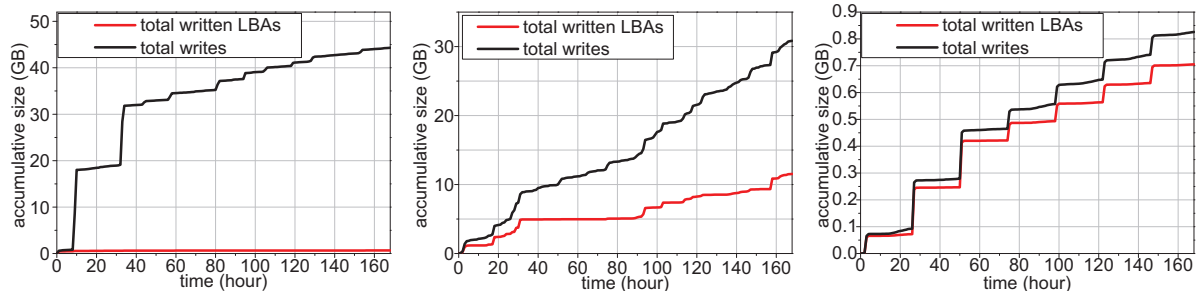
There often exist two levels of write buffers in SSD based computer systems: (1) a host-side buffer that is managed by file system; and (2) a device-side write buffer that is managed by SSD controller. The up-to-date (dirty) data in the write buffers not only prevent write operations from throttling the system performance, but also exploit access locality by adopting LRU replacement policy. Given that the dirty pages in the write buffers are likely to be expunged in short period of time, recent studies proposed to differentiate these pages from other valid pages in the SSDs [10], [11]. These pages are also referred to as *zombie* pages.

We studied the I/O footprint and the total number of write operations in fifteen real world traces that reflect one-week enterprise server operations. The experimental details are in Section 4.1. The footprint is represented using the cumulative number of written LBAs (logical block addresses). Of the fifteen workloads that we tested, nine workloads are similar to Figure 1(a), five workloads are similar to Figure 1(b), and only one workload, i.e., $web2$ is as Figure 1(c). For Figure 1(a), the written LBA space grows very slowly such that most write operations fall in the pages in the write buffers. When a greedy GC (the one in the baseline system) is invoked, many valid pages in the victim block would be zombie/shadow pages. For Figure 1(b), the LBA space grows at modest rate. However, we still found phases that the grow of LBA space pauses even though the number of writes grows linearly. That is, most writes in these phases tend to fall in the write buffers. Only in the $web2$ workload (Figure 1(c)), the LBA space grows almost linearly such that the pages in the write buffers have little reuse opportunities.

To exploit the pages in the write buffers to reduce GC overheads, Lee *et al.* [11] proposed to skip copying zombie pages if the write buffers are non-volatile. The skip policy cannot be applied to volatile write buffers as otherwise a failure event would leave the system in inconsistent state. Lee *et al.* [10] proposed to delay garbage collecting flash blocks if they contain many zombie pages. The design was proposed for systems with large write buffers, e.g., 4GB RAM for a 12GB RAM system [10]. For the mainstream flash based systems, the delay policy tends to lose its effectiveness, as shown in our preliminary study next.

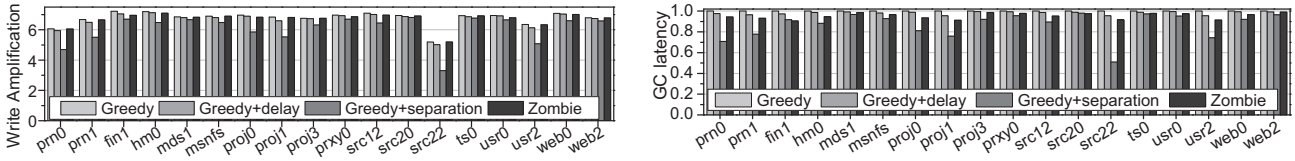
Figure 2 compares the baseline Greedy GC with (1) `Greedy+delay`, (2) `Greedy+separation`; and (3) `Zombie` [10] schemes. `Greedy+delay` is the scheme developed on top of the baseine. It delays the selection of blocks with many zombie pages as victim blocks. `Greedy+separation` is the scheme that enhances the greedy GC by placing zombie pages in dedicated blocks. `Greedy+separation` adopts the same greedy strategy as that in the baseline in determining victim blocks, i.e., choosing the block with the least number of valid pages. `Zombie` is the scheme in [10], which enhances `Greedy+separation` by delaying the selection of blocks with many zombie pages. From the figure, while placing zombie pages to dedicated blocks is an effective mechanism for performance improvement, the delay policy tends to degrade the performance. This is because, for workloads with good access locality, the blocks that are likely to be chosen as victim blocks contain many zombie pages. Delaying choosing such blocks results in choosing the blocks with more valid pages, resulting in larger page copying overhead.

To summarize, while it is beneficial to differentiate zombie/shadow pages from other valid pages in SSDs, we need to develop novel schemes to fully exploit the potentials.



(a) src12 (b) prn1 (c) web2

Figure 1. Compare I/O footpritn (i.e., the LBA size) and the total number of writes in representative traces.



(a) Write amplification (b) GC latency

Figure 2. Separating zombie pages is effective for performance improvement.

3. Cooperative Garbage Collection with Multi-level Buffer

In this section, we present ShadowGC, a novel GC design for mitigating GC induced write amplification and latency issues. We first present an overview of the design and then elaborate its details and analyze its overhead.

3.1. System Overview

Figure 3 presents the SSD based storage system that has ShadowGC embedded in the SSD controller. There exist two write buffers: a host-side write buffer and a device-side write buffer. ShadowGC exploits the pages in both buffers for performance improvement. It consists of two components: (1) Shadow state manager, (2) GC operation optimizer.

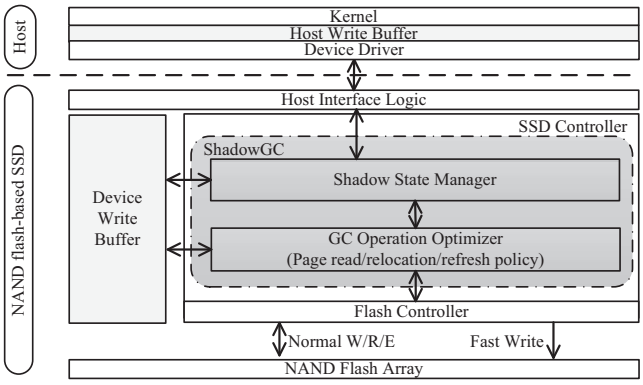


Figure 3. System Overview with ShadowGC.

The shadow state manager in ShadowGC tracks the runtime states of all device pages to assist the GC operation optimizer in enhancing the performance of garbage collection. In particular, it tracks if the corresponding logic page, i.e., the logic page saved at the tracked device page location, has dirty copies residing in one or both of the write buffers. That is, if these device pages are shadows. When relocating valid pages from a victim block at garbage collection time, ShadowGC checks their runtime states and adopts different relocation strategies — for the shadow pages that have dirty copies in the device-side buffer, the GC operation optimizer in ShadowGC extracts the page content from the write buffer, which merges the buffer write back operation with the GC page relocation operation. For the shadow pages that have dirty copies in the host-side buffer, the GC operation optimizer relocates them to dedicated blocks and uses fast write mode to program them.

Since the pages programmed with fast write mode tend to have short retention time, ShadowGC keeps two refresh queues: one to track blocks written in normal write mode; the other for those written in fast write mode. The latter adopts a smaller refresh interval to prevent potential data corruption problems.

ShadowGC minimize its runtime overhead by avoiding changes to the effective policies in the baseline. For example, it adopts the greedy victim block selection, which prevents choosing pages that have more valid pages. While we exploit the pages saved in the write buffers, we keep the buffer replacement algorithm the same as the that in the baseline. We may adopt the traditional LRU (least recently used) algorithm or the state-of-the-art flash-aware buffer replacement algorithms such as CFLRU [12], PTLRU [13] and BPLRU [14].

3.2. Shadow State Manager

To differentiate flash pages that may have dirty copies in different write buffers, the shadow state manager in ShadowGC attaches a 3-bit flag to each flash page to track its state at runtime. As a comparison, a traditional GC uses 1-bit flag to differentiate two types of flash pages, i.e., valid/free and invalid pages. ShadowGC refines the valid/free state to four different states as follows.

- 000: invalid pages;

- 011: host-shadow pages, i.e., the valid pages with dirty copies in the host-side write buffer;
- 101: device-shadow pages, i.e., the valid pages with dirty copies in the device-side write buffer;
- 111: twin-shadow pages, i.e., the valid pages with dirty copies in both write buffers;
- 001: other valid or free pages.

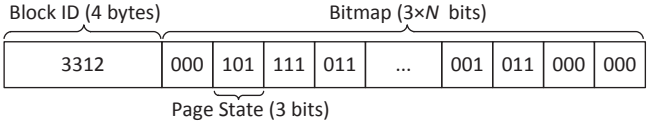


Figure 4. The bitmap for page states in the shadow state manager.

The 3-bit flag tracks a mix of the device page state (i.e., if the page is valid or not) and logical page state (i.e., if the page has shadow copy). A valid device page is labeled as being in shadow state if its corresponding logical page has dirty copies residing in one or both write buffers. ShadowGC stores the states of all pages in a shadow table. As shown in Figure 4, a shadow entry stores the metadata of one block, which consists of one 4B block ID and 3×N-bits pages state information, where *N* is the number of pages in each block.

When logic flash pages are updated in the device side write buffer, ShadowGC updates the shadow table accordingly, with the assistance of FTL. When logic pages are inserted to or expunged from the host side write buffer, ShadowGC passes their logical addresses to the SSD controller by embedding the information within the following I/O read/write commands, which incurs no additional messages. For example, the serial ATA (SATA) 2.6 interface has 5B reserved/unused fields that may be utilized to embed the notification.

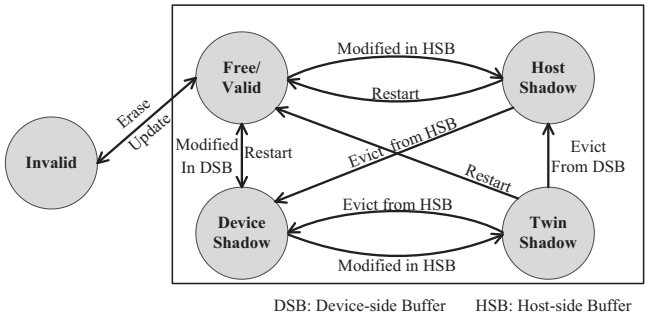


Figure 5. The state transition diagram in ShadowGC.

Figure 5 illustrates the state transition in ShadowGC. For a valid flash page, whenever a write request update its data in the host buffer, the state becomes host-shadow. If it is evicted from host buffer and added into the device buffer, the state changes to device-shadow. If an update request arrives again, which means the flash page is cached in both host buffer and device buffer, the state becomes twin-shadow. When storage system restarts, all three shadow states return to valid. Note that when a twin-shadow page is evicted from device buffer, it is simply discarded without flushing even if it is dirty for the reason that the up-to-date data is still cached in the host buffer.

3.3. ShadowGC Details

The GC operation optimizer enhances the read, program, and refresh operations performed during garbage collection. ShadowGC is triggered when the number of clean blocks drops below a certain threshold. Similar as that in the baseline (i.e., the Greedy GC), ShadowGC chooses the block that has the least number of valid pages as the victim block. The selection policy in ShadowGC does not differentiate the refined states of valid pages. We then discuss how to process the chosen victim block.

Page read policy. ShadowGC needs to relocate the valid pages in the victim block before erasing it. It reads the page content from different locations based on the state of the valid page. For the pages that have dirty copies in the device-side write buffer, i.e., the valid page is in either device-shadow or twin-shadow state, ShadowGC reads the data directly from the write buffer and clears the page's dirty bit in the write buffer. Otherwise, it reads the data from the victim block and uses the ECC to correct read errors (if any), the same as the baseline.

Reading page content from the device side write buffer exhibits two advantages: (i) it merges the GC page copy operation and write back operation of the write buffer. ShadowGC converts the dirty copy in the write buffer to clean page such that expunging it at a later time does not incur additional device write operations. This helps to mitigate the write amplification issue in GC. (ii) since reading the data from the write buffer does not need expensive ECC operation, the read latency can be effectively reduced. This helps to mitigate the GC latency overhead.

Page relocation policy. ShadowGC then relocates the valid pages in the victim block to other blocks. It keeps two active blocks: one is to hold relocated pages from the victim block when these pages have dirty copies in the host-side write buffer, i.e., they are in either host-shadow or twin-shadow states; the other active block is to receive normal writes and other relocated valid pages. As our preliminary study show, page separation is a very effective mechanism in mitigating the GC overhead.

In this paper, we further reduce GC overhead through fast write operations. Flash programming widely adopts the incremental step pulse programming (ISPP) strategy. By using a large program step size ΔV_p in ISPP, the write speed can be improved at the cost of reduced guard band between two states, leading to retention time reduction (relaxation). We exploit that fact that shadow pages tend to have short lifetime and thus relocating them can use fast write mode that have short retention time. Note, the device-shadow pages are relocated to normal active blocks as the up-to-date data have been merged.

By adopting the greedy strategy in victim block selection, ShadowGC minimizes the number of page copies at garbage collection time. Adopting fast write mode further shortens the page copy latency, which effectively mitigates the latency issue in GC. Since the dedicated blocks only contain shadow pages, most pages have become invalid when they are to be reclaimed.

Page refresh policy. As the blocks in the dedicated blocks are programmed using fast write mode, ShadowGC needs to refresh these blocks if they are not reclaimed before the retention threshold. Usually, the dedicated blocks are more likely to have more invalid pages soon and to be selected as victim for cleaning by ShadowGC. As a result, they are unlikely to be refreshed. The only concern is the restart of the storage system. After restart, the write buffer is reset and the shadow pages return to valid, which means the former hot dedicated blocks may become cold and the refresh will be triggered in this case. For this purpose, ShadowGC employs an extra refresh queue to hold blocks written using fast write mode. Our experiments observe that the extra refresh overhead is below 1%, where the impact is negligible.

ShadowGC is designed to survive system crashes and unexpected power failures. Assuming the system may lose power for a duration of D, and the relaxed block retention time is T, we refresh pages written in fast write mode for every $T-D$ interval. When restarting the system, we refresh such pages to prevent data corruption.

3.4. Overhead Analysis

ShadowGC introduces three types of overhead: firmware overhead, storage overhead, and computation overhead. For the firmware overhead, since we add a dedicated block list for relocating host-shadow and twin-shadow pages, we demand two active blocks, i.e., the integration of multi-streamed technology if it is not embedded already. Kang et al. showed that the overhead is negligible [15].

For the storage overhead, as we introduce three new page states to indicate if a page is cached in the host-side buffer or device-side buffer, we need a 3-bit flag for each page, while originally only 1-bit is needed. For a 128GB SSD with the page size of 16KB [16], we need 2MB more storage capacity, which is only additional 0.78% of 256MB RAM capacity on state-of-the-art SSDs. The additional storage requirement with 2MB is small compared to the size of mapping table for FTLs, and to the available RAM of modern SSD. Thus, the storage overhead is negligible.

For the computation overhead, each time when a new page is placed in the write buffer or a least-recently-update page is evicted, we need to find its current physical address from flash translation table and update its page state in the shadow state manager. As the translation table is usually an associative array, we can find the physical address of logical page i at offset i of the array, where the complexity is O(1).

4. Experiment and Analysis

4.1. Experimental setup

To evaluate the effectiveness of the proposed ShadowGC scheme, we used an event-driven simulator [17] to simulate a 1TB MLC NAND flash based SSD. Each block has 512 pages while each page is of 16KB. The default over-provisioning factor of this flash memory is set as 7%. It takes 120 μs to read a flash page, and 300 μs and 600 μs to program a flash page using fast program mode and normal program mode, respectively [19]. We configured the SSD to match those in previous studies and the state-of-the-art storage systems [19], [23]. According to previous studies, the ratio of the write buffer to flash array approximately ranges from 0.1% [21], [23] to 10% [20]. We conservatively chose 128MB and tested write buffers of other sizes. The results for large write buffers are not included in the paper due to page limit. It takes 20 ns to read a page in the DRAM write buffer [22].

In the experiments, we implemented and compared the following garbage collection schemes: (1) `GGC`. This is the baseline scheme that implements the traditional Greedy GC. (2) `ZGC`. This is the scheme that implements the Zombie-Aware GC [10], which delays the selection of flash blocks with many zombie pages (having dirty copies in the host-side write buffer). (3) `HGC`. This is the ShadowGC scheme that is developed on top of `GGC`. For evaluation purpose, we only exploit the pages in the host-side write buffer. That is, the device-side write buffer is used the same as that in the baseline. (4) `DGC`. This is the ShadowGC scheme that is developed on top of `GGC`. It only exploits the pages in the device-side write buffer. That is, the host-side write buffer is used the same as that in the baseline. (5) `SGC`. This is the proposed ShadowGC scheme with pages from both write buffers exploited for performance improvement.

We used the traces from the MSR Cambridge traces and UMass Trace repository. These traces are widely used for studying SSD performance [18], [19]. Before each simulation, the SSD is filled with uniform random workloads so that GC may be triggered for evaluation.

4.2. Efficiency on Garbage Collection

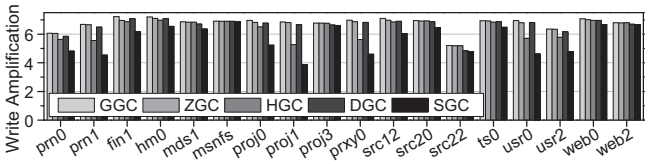


Figure 6. Comparing write amplification in different schemes.

Fig. 6 compares the GC-induced write amplification in different schemes. From the figure, ZGC, HGC and DGC reduce the average write amplification by 0.9%, 6.4% and 2.3% respectively over Greedy GC, while ShadowGC achieves 16.2% reduction. ShadowGC achieves the largest reduction, i.e., 43.4%, in *proj1*, and the smallest reduction, i.e., 0.6%, in *msnfs*. The write amplification reduction comes from: (1) Due to the improved ratio of clean pages in the buffer, the number of writes for eviction is reduced in both DGC and ShadowGC schemes; (2) Since the dedicated blocks that hold relocated host-shadow and twin-shadow pages tend to be invalidated soon, the average number of page copy operations is reduced in ZGC, HGC and ShadowGC schemes.

Fig. 7 compares the accumulative GC induced latency in different schemes, with the results normalized to Greedy

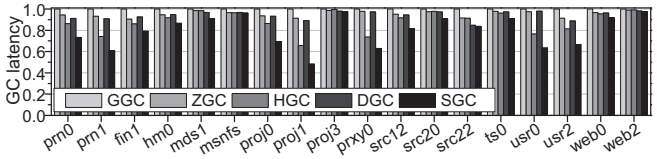


Figure 7. Comparing write GC latency in different schemes.

GC. ShadowGC achieves 20.5% latency reduction compared to Greedy GC. The improvement comes mainly from: (1) Due to the device-shadow and twin-shadow pages, which are read from buffer directly without ECC, the read latency of these flash pages is reduced. (2) Due to the host-shadow and twin-shadow pages, which are programmed quickly with a large program step size, the program latency is reduced.

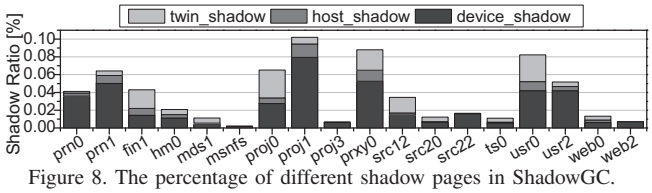


Figure 8. The percentage of different shadow pages in ShadowGC.

To fully understand the performance improvement in ShadowGC, Fig. 8 reports the percentage of different shadow pages. By comparing with the write amplification reduction in Fig. 6 and the GC latency reduction in Fig. 7, we observed that the more shadow pages there are, e.g., *proj1*, the larger improvement ShadowGC achieves.

5. Conclusion

In this paper, we proposed ShadowGC, a novel GC design that exploits the pages in both host-side and device-side write buffers and adopts different read and write strategies to minimize the GC overhead. When garbage collecting flash pages that have dirty copies in the device-side write buffer, ShadowGC reads data from the write buffer such that expunging these cleared pages at a later time introduces no additional writes. When garbage collecting flash pages that have dirty copies in the host-side write buffer, ShadowGC moves them to dedicated blocks and speeds up the movement with fast-write operations. Experimental results show that ShadowGC outperforms its competitors.

Acknowledgment

This work is supported in part by the National Key Research and Development Program of China under grant 2016YFB1000303, in part by the Joint Research Fund for Overseas Chinese, Hong Kong and Macao Young Scientists of the National Natural Science Foundation of China under Grant 61628210, in part by National Science Foundation of USA under Grant CCF-1718080, and in part by the National Natural Science Foundation of China under grant 61672423, and in part by the Key Science and Technology Program of Shaanxi Province, China under Grant 2016SF-428.

References

[1] J. Huang, A. Badam, L. Caulfield, S. Nath, S. Sengupta, B. Sharma, and M. k. Qureshi, "FlashBlox: Achieving Both Performance Isolation and Uniform Lifetime for Virtualized SSDs," *FAST*, pp. 375–390, 2017.

[2] Y. Lee, J. Kim, S. Lee, and S. Maeng, "Exploiting Sequential and Temporal Localities to Improve Performance of NAND Flash-Based SSDs," *ACM Transactions on Storage*, vol. 12, no. 3, pp. 15, 2016.

[3] Y. Cai, G. Yalcin, O. Mutlu, E. Haratsch, A. Cristal, O. Unsal, and K. Mai, "Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime," *ICCD*, pp. 94–101, 2013.

[4] M. Shafaei, D. Peter, and F. Jim, "Write amplification reduction in flash-based SSDs through extent-based temperature identification," *HotStorage*, 2016.

[5] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann, "The Devil Is in the Details: Implementing Flash Page Reuse with WOM Codes," *FAST*, pp. 95–109, 2016.

[6] G. Yadgar, E. Yaakobi, and A. Schuster, "Write Once, Get 50% Free: Saving SSD Erase Costs Using WOM Codes," *FAST*, pp. 257–271, 2015.

[7] J. Guo, C. Min, T. Cai, and Y. Chen, "A design to reduce write amplification in object-based NAND flash devices," *CODES+ISSS*, pp. 1–10, 2016.

[8] H. Chang, C. Ho, Y. Chang, Y. Chang, and T. Kuo, "How to enable software isolation and boost system performance with sub-block erase over 3D flash memory," *CODES+ISSS*, pp. 6, 2016.

[9] M. Jung, W. Choi, S. Srikantaiah, J. Yoo, and M. Kandemir, "HIOS: A host interface I/O scheduler for solid state disks," *ACM SIGARCH Computer Architecture News*, vol. 42, no.3, pp. 289–300, 2014.

[10] Y. Lee, J. Kim, S. Lee, and S. Maeng, "Zombie chasing: Efficient flash management considering dirty data in the buffer cache," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 569–581, 2015.

[11] E. Lee, J. Kim, H. Bahn, and S. Noh, "Reducing Write Amplification of Flash Storage through Cooperative Data Management with NVM," *MSST*, 2016.

[12] S. Park, D. Jung, J. Kang, J. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," *CASES*, pp. 234–241, 2006.

[13] J. Cui, W. Wu, Y. Wang, and Z. Duan, "PT-LRU: a probabilistic page replacement algorithm for NAND flash-based consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 60, no. 4, pp. 614–622, 2014.

[14] H. Kim, and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage," *FAST*, pp. 1–14, 2008.

[15] J. Kang, J. Hyun, H. Maeng, and S. Cho, "The Multistreamed Solid-state Drive," *HotStorage*, 2014.

[16] J. Im, W. Jeong, D. Kim, et. al, "A 128Gb 3b/cell V-NAND Flash Memory with 1Gb/s I/O Rate," *ISSCC*, 2015.

[17] P. Desnoyers, "Analytic models of SSD write performance," ACM Transactions on Storage, vol. 10, no. 2, pp. 8, 2014.

[18] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting Intra-Request Slack to Improve SSD Performance," In *ASPLOS*, pp. 375–388, 2017.

[19] J. Cui, W. Wu, X. Zhang, J. Huang, and Y. Wang, "Exploiting latency variation for access conflict reduction of NAND flash memory," *MSST*, 2016.

[20] C. Wang, W. Wong, "TreeFTL: An Efficient Workload-Adaptive Algorithm for RAM Buffer Management of NAND Flash-Based Devices," *IEEE Transactions on Computers*, vol. 65, no. 8, pp. 2618–2630, 2016.

[21] D. Kang, S. Han, Y. Kim, and Y. Eom, "CLOCK-DNV: a write buffer algorithm for flash storage devices of consumer electronics," *IEEE Transactions on Consumer Electronics*, vol. 63, no. 1, pp. 85–91, 2017.

[22] K. Chang, A. Kashyap, H. Hassan, S. Hsieh, D. Lee, T. Li, G. Khan, and O. Mutlu, "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization," *SIGMETRICS*, pp. 323–336, 2016.

[23] Samsung 960 EVO 1TB M.2 PCIe NVMe SSD Review. https://www.kitguru.net/components/ssd-drives/simon-crisp/samsung-960-evo-1tb-m-2-pcie-nvme-ssd-review/