Delayed TRIM for Reducing Trim Overhead

Jungheon Kim, Dongkun Shin
Department of Electrical and Computer Engineering
Sungkyunkwan University
Suwon, Korea
{kjh990705, dongkun}@skku.edu

***Abstract*—We propose Delayed TRIM to mitigate the overhead associated with TRIM requests. While conventional TRIM reduces unnecessary data copying in GC by instantly notifying the device of deleted data, it incurs considerable overhead. In contrast, Delayed TRIM executes only the essential steps immediately and defers the remaining actions until the device is idle. A trim buffer and trim validity bitmap are employed to ensure data integrity during this delay. Experiments with DaisyPlus OpenSSD demonstrate that this technique maintains the WAF while effectively reducing TRIM overhead.**

***Index Terms*—Solid State Drives, Flash memory, Storage system, Flash Translation Layer, TRIM, DISCARD Command**

I. Introduction

A TRIM request allows the host to notify the device of deleted data, enabling the Flash Translation Layer(FTL) to mark pages as invalid and reduce unnecessary copying during Garbage Collection(GC). However, TRIM request is treated like I/O operations and have an overhead that increases with the size of the TRIM range (Saxena and Swift[1]). To mitigate repetitive TRIM requests, some file systems, such as F2FS[2], delay these requests until the device is idle. However, the device is not informed that TRIM requests are being delayed, so it repeats unnecessary copy operations during GC. Other research, such as iDiscard[3], reduces the Write Amplification Factor(WAF) by enabling TRIM requests for ghost pages in EXT4, but it does not address the overhead associated with TRIM requests. TRIM requests involve three steps.

- Step 1. Receiving the request and storing data via DMA
- Step 2. Invalidating data in the write buffer
- Step 3. Invalidating pages in the L2P and P2L tables

Steps 2 and 3 account for 80% of the TRIM processing time.

We propose Delayed TRIM, which executes step 1 immediately and defers steps 2 and 3 until the device is idle. Additionally, if an I/O request occurs during Delayed TRIM processing, I/O request is processed first to minimize the overhead of TRIM request.

II. Delayed TRIM

A. Delayed TRIM Scenario

Delayed TRIM, which defers TRIM request, can cause several problems. Figure 1 illustrates two potential issues associated with Delayed TRIM. The first issue is that new write requests can occur in the region where the TRIM operation is delayed. For instance, a TRIM request is made at t1, and subsequent write requests occur at t2 and t3 in the same area. If the TRIM request from t1 is processed later at t4, when the device is idle, the data written to LBA 100 and 101 at t2 and t3 might be erroneously invalidated by the delayed TRIM, even though it is valid data. The second issue arises if GC occurs before the Delayed TRIM processing, as shown at t7 in Figure 1. In this scenario, blocks that should have been invalidated, such as LBA 200 and 201, may still be marked as valid, leading to unnecessary copy operations during GC.

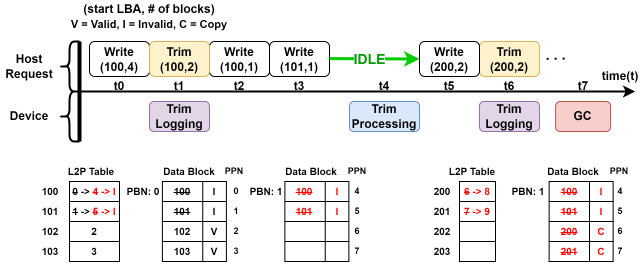

Fig. 1. Delayed TRIM Scenario

B. Processing DelayedTRIM

Delayed TRIM utilizes a trim buffer to store TRIM requests and a trim validity bitmap to track the logical pages slated for TRIM. Upon receiving a TRIM request, Delayed TRIM records the starting LBA and the number of blocks to the trim buffer, and updates the trim validity bitmap to reflect these logical pages (Step 1). The remaining steps are then deferred. The trim validity bitmap can be managed either as a full bitmap (approximately 8 MB per 1 TB SSD) or as a partial bitmap, which dynamically allocates smaller bitmaps to reduce memory usage. If a write request occurs when the TRIM operations are delayed, the trim validity bits corresponding to the logical pages that overlap with the write request range in the trim validity bitmap are invalidated to prevent incorrect TRIM operations. When the device becomes idle, the system traverses the trim buffer and processes the remaining steps, which involve invalidating the write buffer, L2P, and P2L tables (steps 2 and 3). During this time, the trim validity bitmap is used to verify the validity of the TRIM operations.

If an I/O request arrives during Delayed TRIM processing, the current TRIM context is saved, and the TRIM process is paused. If a GC occurs during the TRIM delay, the Delayed TRIM is executed before the GC. During this period, even if an I/O request arrives, the Delayed TRIM is completed first.

979-8-3503-8855-8/24/$31.00 ©2024 IEEE
NVMSA.2024

Additionally, We can manage the TRIM overhead caused by GC by adjusting the amount of TRIM performed before GC. This is covered in the Evaluation section.

III. EVALUATION

In this study, Delayed TRIM is implemented by modifying the DaisyPlus OpenSSD source code. The DaisyPlus OpenSSD is the latest board from the OpenSSD project and is widely used in recent research[4]. This board is equipped with two 256GB NAND Flash memory modules and 2GB of DRAM; however, for our experiments, we utilized only 32GB of the NAND Flash. The experimental environment was set up on Ubuntu 18.04.6 LTS with the Linux 4.15.0-210 kernel. Using the FIO benchmark, we performed the experiment after filling 75% of the device's capacity, with three iterations of 32MB and 2GB TRIM operations and 10GB of random writes.

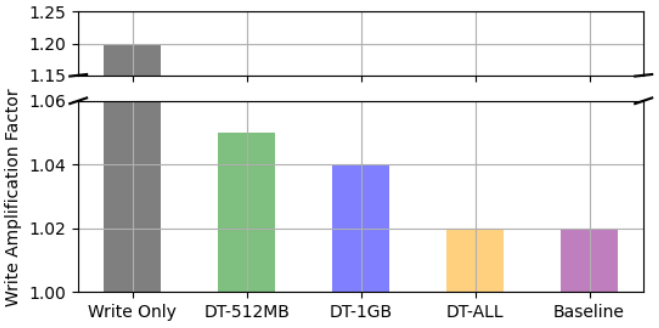

Fig. 2. WAF Comparison

Figure 2 illustrates the WAF results for the baseline (Conventional TRIM), Delayed TRIM, and Write-Only scenarios. We experimented with three Delayed TRIM configurations: DT-512MB (TRIM 512MB), DT-1GB (TRIM 1GB), and DT-ALL (TRIM ALL), based on the amount of TRIM executed before the GC. These configurations are the same in Figure 3 and 4. As We can see in Figure 2, The Write-Only scenario exhibited the highest WAF at approximately 1.2. This was followed by DT-512MB at around 1.05, DT-1GB at about 1.04, and both the Baseline and DT-ALL scenarios at roughly 1.02.

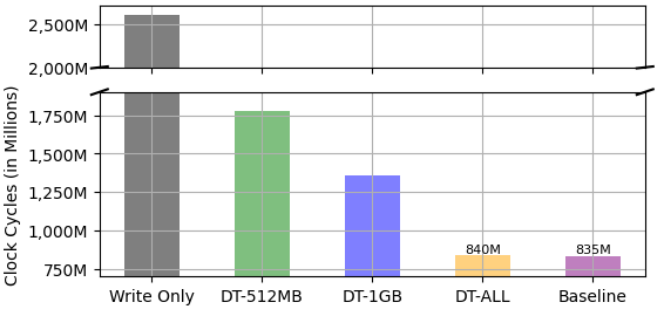

Fig. 3. GC Latency Comparison

Figure 3 shows the GC latency results. The Baseline scenario exhibited the lowest GC latency. For DT-ALL, the latency of the GC itself matched that of the Baseline, as all TRIM operations were applied. However, the overall GC latency for DT-ALL was slightly higher than the Baseline because all delayed TRIM operations were executed immediately before the GC. In contrast, DT-1GB and DT-512MB showed an significant increase in total GC latency compared to the Baseline, attributable to more frequent GC operations.

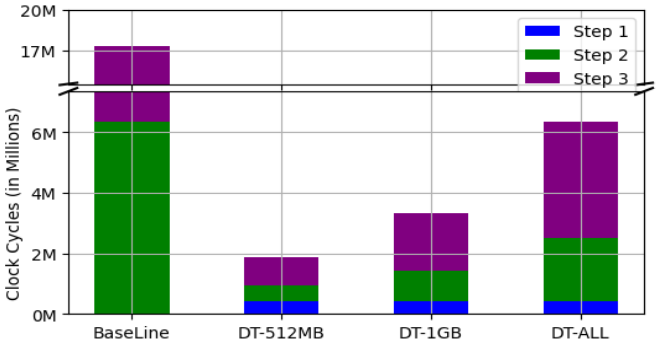

Fig. 4. Trim Overhead Breakdown

Figure 4 shows the TRIM overhead for each TRIM scenario. The overhead is calculated by excluding the overhead of TRIM operations processed during idle states. It is evident that Step 1 of Delayed TRIM, which includes processing the trim buffer and trim validity bitmap, results in increased overhead compared to the Baseline. However, since TRIM operations are executed when the device is idle, the TRIM overhead is reduced to 36% for DT-ALL, 20% for DT-1GB, and 10% for DT-512MB compared to the Baseline.

From Figures 2, 3, and 4, we can see that Delayed TRIM has similar WAF performance to Baseline and can reduce the TRIM overhead. Also, By adjusting the amount of TRIM that is performed before GC, we can manage the trade-off between GC and TRIM overhead. This balance can help us to optimize performance and reduce unnecessary copy operations.

IV. CONCLUSION

We propose Delayed TRIM to reduce the overhead of conventional TRIM. We analyzed the overhead of each step in the TRIM process. This approach maintains WAF performance with delaying most processing steps that contribute the most to overhead to be processed when the device is idle.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2021R1A2C2095125)

REFERENCES

[1] M. Saxena and M. M. Swift, "FlashVM: Virtual memory management on flash," in Proc. USENIX Annu. Techn. Conf. (ATC), 2010, p. 14.
[2] C. Lee, D. Sim, J. Y. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in Proc. 13th USENIX Conf. File Storage Technol. (FAST), 2015, p. 273
[3] D. H. Kang and Y. I. Eom, "iDiscard: Enhanced Discard() Scheme for Flash Storage Devices," 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), 2018, p. 360.
[4] Jaewook Kwak, Sangjin Lee, Kibin Park, Jinwoo Jeong, and Yong Ho Song. 2020. Cosmos+ OpenSSD: Rapid Prototype for Flash Storage Systems. ACM Trans. Storage 16, 3, Article 15 (August 2020), 35 pages.