Host-Driven Block Management Scheme Using Log-Structured File System for Memory-Constrained SSD

Sihoon Choi, Hyunho Gwak, and Dongkun Shin

College of Information and Communication Engineering,
Sungkyunkwan University, Suwon, Korea
beswan@skku.edu, gusghrhkr@skku.edu, dongkun@skku.edu

Abstract **–** Recently, Solid-State Disks (SSDs) are widely adopted by high-performance storage systems. As the capacity of SSD increases, the DRAM size for address mapping table becomes a critical problem of SSD. To address this problem, we propose a novel log-structured file system for block-level maping flash translation layer. By synchronizing the address spaces of file system and flash translation layer, the scheme can eliminate the duplicated garbage collection and can minimize the flash block management cost. In experiments with a real SSD device, the proposed scheme improved the write performance significantly for the random write intensive workloads.

Keywords: Flash memory, File system, Flash Transaltion Layer, Solid-State Disk.

1 Introduction

Recently, Solid-State Disks (SSDs) are widely adopted by high-performance storage systems. SSD uses flash memory as storage media. A flash memory chip is composed of several blocks, and a block is composed of several pages. While the erase operation is performed by the unit of block, the program or read operation is performed by the unit of page.

Flash memory does not support in-place update due to its “erase-before-write” constraint. Therefore, several out-of-place update schemes are used by SSD firmware, called flash translation layer (FTL), which can emulate a normal block device for flash memory. In order to handle the out-of-place update scheme, FTL manages an address mapping table that provides the translation information between a logical address and a physical address. Generally, recent SSDs use the page-level address mapping rather than the block-level address mapping since the former provides a higher performance.

The size of mapping table is increasing in proportion to the capacity of SSD. For example, a 1TB size of SSD requires 512MB of DRAM space for mapping table if it uses 8KB of page-level mapping. However, a large capacity of DRAM requires extra power consumption, space, and hardware cost. Since the capacities of SSDs are exponentially increasing, the page-level mapping FTL can be no longer adopted by the large capacity of SSDs.

Several techniques are proposed for reducing the memory space of address mapping table in SSD. In DFTL [1] scheme, only a portion of the page-level mapping table is cached at DRAM while the overall mapping table is stored in flash memory. If the mapping entries that are associated with an incoming I/O request are not cached, DFTL replaces old mapping entries with the required mapping entries on demand. The replacement operation causes extra flash memory accesses, and thus degrades SSD performance. In particular, when the storage access pattern is random, there will be frequent replacement operations. The block-level mapping with several log blocks [2] can reduce the mapping table size. However, the log block merge operation invokes a large performance overhead. The block-level mapping also shows a poor performance when the workload is random since the random writes increase the log block merge cost. As long as the poor random I/O performances of these alternative mapping techniques are not solved, they cannot be solutions for the mapping table size problem.

In order to mitigate the random I/O performance problem of the block-level mapping FTL, the log-structured file system (LFS) can be useful since it can control the write access pattern due to its out-of-place update scheme. When a file system block should be updated, LFS does not overwrite the block, instead it writes new data at another free block. Therefore, LFS can determine the write pattern considering the internal cost within SSD. However, LFS must perform the garbage collection in order to reclaim invalid blocks. The garbage collection copies valid blocks in the victim segment to the free segment. A critical problem of garbage collection is that the same operations will happen at SSD. Even though an LFS performs the garbage collection to make an empty segment, FTL will also perform the garbage collection to make a free flash memory block. The duplicated garbage collections reduce the I/O performance and lifespan of SSD [3].

To solve the duplicated address space management problem, Lee *et al*. [4] proposed the REDO scheme which is composed of a simple block-level mapping FTL and a LFS. The garbage collection is performed only by the file system. However, the host-level garbage collection invokes many data transfer operations between host and device. In addition, if the file system block size and the flash memory page size are different, the host file system cannot perfectly control the internal behavior of SSD. For example, if SSD uses 8KB of page size but LFS uses 4KB of block size, SSD should perform the read-modify-write operation for 4KB write request.

In this paper, we proposes an integrated file system and FTL scheme, which is composed of a block-level mapping FTL for small mapping table and a LFS. In order to remove the data transfer operations during garbage collection, our LFS uses the threaded-logging scheme, which reclaims invalid blocks (holes) by overwriting instead of copying then into other free segments. The LFS selects the holes considering the write cost in SSD.

2 Archtecture

2.1 Block-Level Mapping FTL

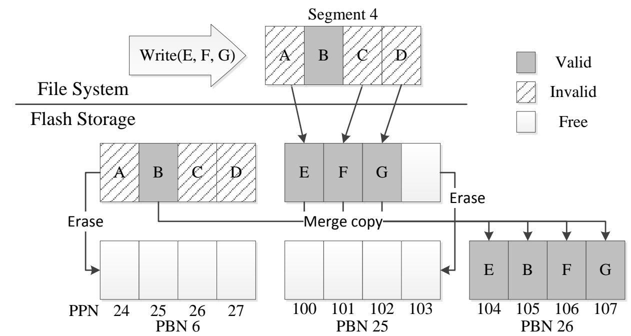
Our scheme uses the block mapping, which allocates a replacement block to update a flash block. The replacement block uses the in-place write scheme, i.e., the physical location within a block of each logical page is determined based on the logical address. Therefore, the data location can be directly controlled by the host file system which determines the logical address. The FTL is called as S-FTL.

In the block mapping scheme with out-of-place log blocks [2], the log block is programmed in the order of request arrival time. Therefore, for random write requests, there are significant log block merge overhead as shown in Figure 1(a). However, our FTL performs the page padding technique in order to maintain the in-place write scheme as shown in Figure 1(b).

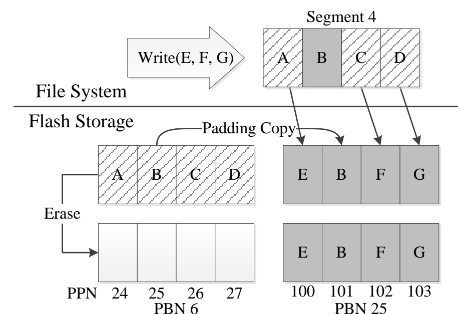
2.2 LFS

Our LFS has several features. First, the LFS segment size is equal to the flash memory block size of SSD. Second, only the threaded-logging scheme is used to remove the garbage collection cost. When the threaded-logging selects holes for new data, it selects the block location considering the padding cost. Third, the threaded-logging utilizes the holes of a segment at the increasing address order to generate only sequential write requests for the associated flash block. The file system is called as S-LFS.

Figure 2 (a) shows an example of when the LFS segment size is smaller than the flash block size. Although the LFS considers the segment 1 as a free segment, the segment 1 is associated with a part of flash block. Therefore, the file system cannot directly control the padding cost within SSD. As shown in Figure 2 (b), if the file system segment size and the flash block size are identical, there are no unexpected page copy operations in SSD.

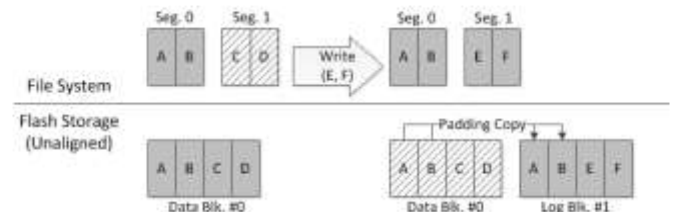


(a) Block mapping w/ Log block

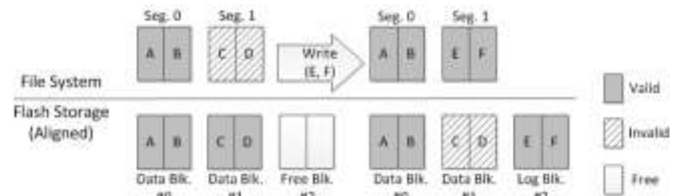


(b) Block mapping w/ in-place replacement block

Figure 1. FTL operation comparison



(a) Segment and flash block is unaligned



(b) Segment and flash block is aligned

Figure 2. FTL behavior comparison according to the alignment between the segment and the flash block

When the LFS generates the write requests in the threaded-logging manner, if the block size of file system and the flash page size in SSD are different, there will be read-modify-write operations in FTL and the in-place write policy in a flash block can be broken. To solve this problem, our LFS performs the host-level padding

technique if the hole size is less than the flash page size or the hole location is unaligned to the flash page size. It reads valid blocks from SSD, merges it with new data, and sends aligned write requests to SSD.

3 Experiments

We implemented the proposed LFS technique at F2FS [5]. The block-level mapping FTL was implemented at the Jasmine OpenSSD platform [6]. For comparisons, we also implemented DFTL [1] and the pure block mapping of REDO [4] at the OpenSSD platform. The ext4 file system is used for DFTL, and an F2FS-based LFS is used for REDO. The flash memory chips in OpenSSD uses 16KB of pages and 128 pages of block. The total capacity of OpenSSD is 3GB and the overprovision ratio is 3%. Several benchmark programs are used such as Tiobench, TPC-C, and Filebench (varmail workload). The target SSD is initialized with dummy data such that the garbage collections are invoked during the target workload executions.

As shown in Figure 3, the proposed scheme improves the performance by 71%~793%. The ext4 with DFTL shows significantly low performance due to the random write patterns of workloads. REDO performs well in the Tiobench workload since REDO changes the random write requests of the benchmark into sequential write requests. However, for other benchmarks, REDO shows the lowest performance due to its unaligned write requests. Therefore, as shown in Figure 4, REDO suffers from the full merge operation overhead when small random write requests are dominant. Figure 4 shows that the proposed scheme performs only the switch merges and the partial merges since the replacement block is programmed by the in-place scheme.

4 Conclusions

In this paper, we proposed a log-structured file system for block-level mapping flash translation layer, which can reduce the mapping table size of SSD. By the in-place write scheme for the flash blocks, it can avoid high-cost of block merge operations. In order to eliminate duplicated garbage collections, the LFS uses only threaded-logging scheme. Experiments showed that the proposed scheme improves the write performance significantly for random write intensive workloads. It can also extend the lifespan of SSD by 72% on average.

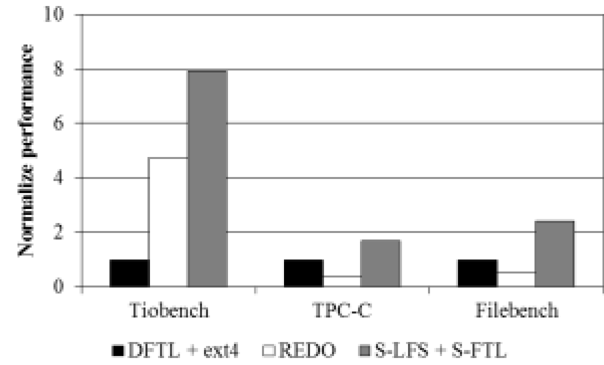


Figure 3. Normalized write performance

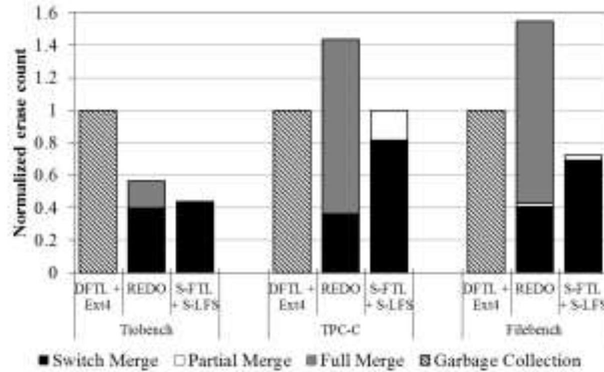


Figure 4. Normalized erase count of each FTLs

References

[1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings", In Proc. of the International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), pp. 229–240, March 2009.

[2] J. Kim, J.M. Kim, S.H. Noh, S. Min, and Y. Cho. "A Space-Efficient Flash Translation Layer for Compactflash Systems", IEEE Transactions on Consumer Electronics, 48(2):366–375, May 2002.

[3] Jingpei Yang, Ned Plasson, Greg Gillis, Nisha Talagala, and Swaminathan Sundararaman. "Don't Stack Your Log On My Log", In Proc. of the USENIX Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW), October 2014.

[4] S. Lee, J. Kim, and A. Mithal, "Refactored Design of I/O Architecture for Flash Storage", Compututer Architecture Letters, pp. 99:1–1, 2014.

[5] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A New File System for Flash Storage", In Proc. of the USENIX File and Storage Technologies (FAST), Fabruary 2015.

[6] The OpenSSD Project, http://www.openssd-project.org