Reducing Garbage Collection Overhead of Log-Structured File Systems with GC Journaling

Hyunho Gwak, Yunji Kang, and Dongkun Shin
College of Information and Communication Engineering
Sungkyunkwan University
Suwon, Korea
gusghrhkr@skku.edu, oso41@skku.edu, dongkun@skku.edu

***Abstract*—The log-structured file system (LFS) writes all modifications to storage sequentially with append-only logging scheme. This characteristic of LFS is very advantageous to flash storages since the flash memory does not permit in-place overwrite. However, LFS has a high garbage collection (GC) overhead. In particular, under the lazy metadata update scheme, each GC process should invoke the high-cost checkpointing which flushes all the dirty metadata and normal data to storage. The long GC latency will degrade the response times of user requests. In this paper, we propose a GC journaling technique, which journals only the file system changes relevant to the GC process without invoking the high cost checkpointings.**

***Keywords*— log-structured file system; flash storage; garbage collection; checkpointing**

I. Introduction

Flash memory has been used widely in various consumer devices such as mobile phones and smart TVs due to its high performance, low power consumption, and shock resistance. In particular, recent mobile devices use embedded multimedia card (eMMC) and secure digital (SD) card, which use flash memory as storage media. These devices include a special firmware, called flash translation layer (FTL), which handles all the idiosyncrasy of flash memory and provides the standard block interface to the host. Since the flash memory does not permit in-place overwrite, it shows poor performance at random write requests.

Since the log-structured file system (LFS) [1] generates only the sequential write requests with the append-only logging scheme, LFS is a suitable file system for flash storages. There are several log-structured flash file systems such as YAFFS2 and UBIFS, which are targeting for pure NAND flash memory chip. However, current log-structured file systems show no good performance for flash memory storages. The first reason is its wandering tree problem. When a write operation updates a data block in LFS, its direct index block and indirect index blocks should be also updated recursively since the location of data block is changed. The second problem is the high garbage collection (segment cleaning) cost of LFS. During the garbage collection (GC) process, all the valid blocks in victim segments should be moved to clean segments.

Recently, a new log-structured file system, called F2FS [2], is proposed, which is designed for FTL-embedded flash memory storage devices such as eMMC and SD card. In order to avoid the wandering tree problem, F2FS uses the lazy metadata update scheme, where the dirty metadata blocks are not flushed immediately. They are flushed during the checkpointing instead. The checkpointing generates a consistent file system recovery point called checkpoint. However, under the lazy metadata update scheme, the garbage collection should invoke the checkpointing in order to record the changes of block locations. After the checkpointing, the cleaned segments can be reused by other write requests. Since all the dirty data and metadata should be written at the storage during the checkpointing, the latency of garbage collection will be increased.

The long latency of garbage collection can affect the response time of user request adversely. In this paper, we propose a *GC journaling* (GCJ) technique that can be used instead of checkpointing during the GC process. GCJ uses a journal space where all the information about the blocks moved by GC is recorded. With the journal, the file system consistency can be guaranteed without checkpointing at system crash. By removing the checkpointing during garbage collection, the proposed GCJ scheme can significantly reduce the garbage collection latency.

II. Garbage Collection Journaling

Fig. 1 shows the process of garbage collection. The valid blocks of B_1, B_2, and B_3 in the victim segment S1 are copied into the free segment S2 in order to make the victim segment as a clean segment. Under the lazy metadata update scheme, the changed in-memory metadata will not be flushed into the storage immediately, and the in-storage file system metadata will refer to the old locations of the moved blocks. If user write requests overwrite the blocks in the victim block, the file system consistency will be broken at system crash. Therefore, the checkpointing should be performed after the garbage collection in order to change the in-storage metadata.

The proposed GC journaling does not perform the checkpointing. Instead, the GC journal blocks are written at the storage. Each entry of journal block includes the original and target block addresses of a valid block that is moved during the garbage collection. For example, in Fig. 1, since B_1 in the block address of 3072 is copied into the block with the block address of 1536, the journal entry has the information. The last entry of journal blocks has the current checkpoint version number. The journal blocks are flushed into the journal space of storage before the victim segment is changed

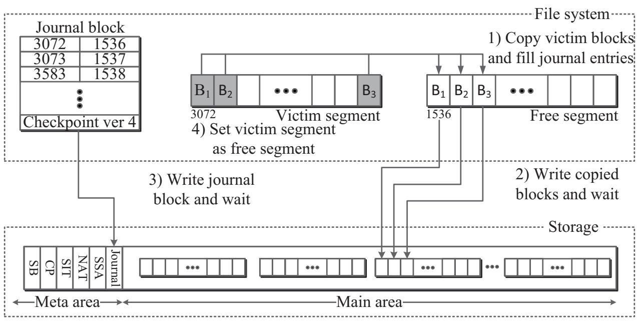
File system
Journal block
3072 1536
3073 1537
3583 1538
Checkpoint ver 4
1) Copy victim blocks and fill journal entries
Victim segment
Free segment
4) Set victim segment as free segment
3) Write journal block and wait
2) Write copied blocks and wait
Storage
SB CP SIT NAT SSA Journal
Meta area
Main area

Fig. 1. The process of segment cleaning with GC journaling.

into a clean segment. Since the GCJ scheme records the changed block addresses of valid data, the reclaimed blocks can be reused by other write requests without checkpointing. If there is a system crash, the file system state can be recovered with the GC journals.

The recovery procedure under GCJ is follows. First, the file system is recovered to the last checkpoint. Second, the file system metadata are changed based on the valid GC journals. If a checkpointing is invoked, all the recorded GC journal blocks are invalidated. Therefore, the remaining valid journal blocks have been written after the last checkpoint. If a block was copied by GC and it was overwritten after the last checkpoint, the in-storage metadata will point to the old block address. However, since the GC journal block has the new block address value, the recovery operation can modify the metadata by applying the file system changes by the garbage collection.

One journal block is 4 KB, and it can contain 511 journal entries. When a segment consists of 512 blocks and a garbage collection cleans one victim segment, only one GC journal block will be written during the segment cleaning. However, the original checkpointing scheme should flush all the dirty blocks. Therefore, the number of flushed blocks can be reduced significantly by the GCJ scheme.

In order to avoid a long recovery time, the maximum number of valid journal blocks is limited. If the number of valid journal blocks exceeds a predefined limit, the GCJ scheme invokes the checkpointing, which invalidates all the journal blocks. In our implementation, the reserved journal space can contain 512 journal blocks, and thus GCJ invokes the checkpointing if the number of valid journal blocks is 512.

III. EXPERIMENTS

We evaluated the effects of GC journaling on two flash storage devices, SSD and SD card. The 256 GB of SSD has 234 MB/s and 122 MB/s of sequential and random write performances, respectively. The 16 GB of MicroSD has 10 MB/s and 3 MB/s of sequential and random write performances, respectively. The proposed scheme is implemented at F2FS. Generally, LFS uses the normal append logging. However, F2FS uses the adaptive logging scheme which uses the normal append logging and the threaded logging [3] selectively. The total execution times of three benchmark programs are measured under four different F2FS configurations: adaptive logging (AL) without GCJ, AL with GCJ, normal logging (NL)

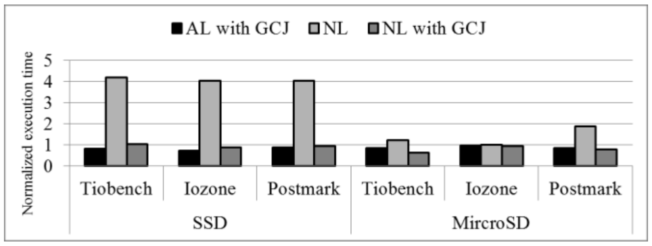

Fig. 2. Normalized execution times under different F2FS configurations.

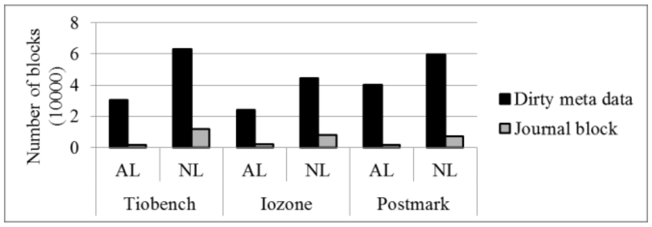

Fig. 3. The number of written blocks during segment cleaning (SSD).

without GCJ, and NL with GCJ. At each experiment, the file system is initialized. First, two thousands number of files are created. Then, the file system utilization becomes 70%. Second, the files are randomly updated in order to make holes in segments until there is only 5% of clean segments. Then, all the following write requests can invoke the garbage collections or threaded loggings.

Fig. 2 shows the execution times of benchmark programs under different schemes. The values are normalized by the execution times under the AL scheme. AL invokes the garbage collections infrequently, and generates random writes instead by the threaded logging. Since SSD has similar performances at random write and sequential write, the AL scheme shows significantly better performance than the NL scheme does. However, AL will show poor sequential read performance due to the fragmented data blocks.

The GCJ reduces the execution times at the NL scheme significantly, since GCJ reduces the cleaning latencies by reducing the number of blocks flushed during the segment cleaning. Considering several problems of the AL scheme, the NL scheme with GCJ can replace the AL scheme. The execution times at the AL scheme are slightly reduced by GCJ, since the AL scheme hardly invokes the garbage collections.

Fig. 3 shows the total number of blocks flushed during the garbage collections. The number of journal blocks at the GCJ scheme is on average only 11% of the number of dirty metadata blocks which are flushed by the checkpointings at the original scheme. Therefore, GCJ can increase the lifetime of flash storage which has a limited program and erase cycles.

REFERENCES

[1] M. Rosenblum and J. K. Ousterhout, “The design and implementation of a log-structured file system,” ACM Transactions on Computer Systems (TOCS), vol. 10.1, pp. 26-52, 1992.

[2] C. Lee, D. Sim, J. Hwang, and S. Cho, “F2FS: A new file system for flash storage,” Proceedings of the USENIX Conference on File and Storage Technologies (FAST), 2015.

[3] Y. Oh, E. Kim, J. Choi, D. Lee, and S. H. Noh, “Optimizations of LFS with slack space recycling and lazy indirect block update,” ACM Proceedings of the 3rd Annual Haifa Experimental Systems Conference, 2010.