fsync-aware Multi-buffer FTL for Improving the fsync Latency with Open-Channel SSDs

Somm Kim[1], Yunji Kang[2], and Dongkun Shin[1]
[1]*Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Korea*
[2]*Department of IT Convergence, Sungkyunkwan University, Suwon, Korea*
Email: {sommkim, oso41, dongkun}@skku.edu

***Abstract*—Open-Channel SSDs are widely studied because of their advantages such as predictable latency, efficient data placement, and I/O scheduling. Currently, the Linux kernel includes pblk (The Physical Block Device), a host FTL that supports Open-Channel SSDs. In addition, there are recent studies that expand the single-threaded architecture of pblk to multi-threaded architecture: MT-FTL and QBLK. However, both pblk and recent studies were designed without considering fsync latency. However, since the fsync system call is performed synchronously, has a great effect on the performance of the system. In this paper, we propose FA-FTL, which is a host FTL considering fsync latency. Experiments show that FA-FTL is 141% higher than pblk and 119% higher than MT-FTL.**

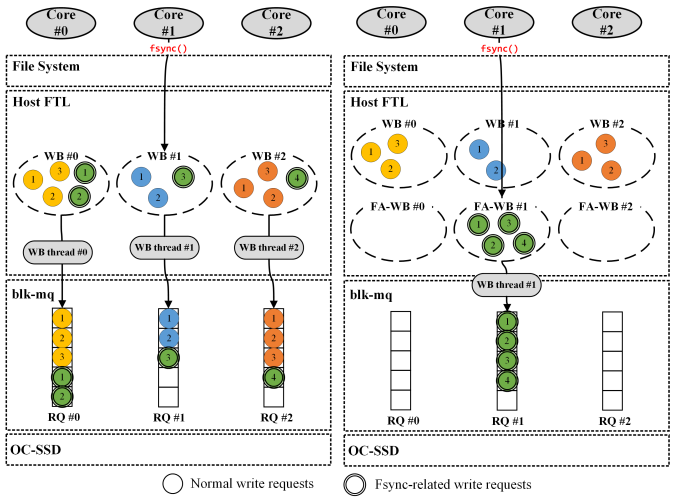
***Index Terms*—Solid State Drives, Flash Translation Layer, Open-Channel SSDs, Host FTL, fsync, FLUSH Command**

I. Introduction

Open-Channel SSDs [1] are a new type of Solid State Drives (SSDs) in which the Flash Translation Layer (FTL) is present on the host. It provides advantages such as predictable latency, efficient data placement and I/O scheduling that cannot provided by firmware based FTL. Linux kernel version 4.12 and later provides a host FTL called pblk. Pblk consists of a write buffer and a writer thread. The write buffer is used to buffer write requests, and the writer thread is used to transfer write requests to the storage in the write buffer. MT-FTL [3] is a host FTL that improves the performance bottleneck caused by pblk's single-threaded architecture. MT-FTL has a write buffer and a writer thread for each core. QBLK [6] is also a host FTL that adopts a multi-threaded architecture instead of a single-threaded architecture. In addition, they applied the channel-based address management instead of line-based address management, and applied lock-free mapping table structure and fine-grained draining.

Although overall I/O throughput is improved in the recent studies, the fsync latency is not considered. Because the fsync system call operates synchronously, it has a significant impact on system performance. We find that there is a point in the design of the existing host FTLs to improve fsync latency. In the existing host FTLs, all write buffers should be flushed during fsync, since the fsync-related data blocks can exist at any write buffer. All write buffers contain data blocks that are not related to fsync, making fsync latency longer. Therefore, in this paper, we adopt a fsync-dedicated buffer to flush only fsync-related data blocks during fsync.

978-1-7281-3854-1/19/$31.00 ©2019 IEEE



(a) MT-FTL (b) FA-FTL

Fig. 1: Comparison of flushing during fsync

II. FA-FTL: Fsync-aware Host-level FTL

A. Design

FA-FTL uses fsync-aware write buffers to buffer fsync-related data blocks. When the fsync system call is invoked over the threshold, FA-FTL regards the file as fsynced file and maps it to a certain fsync-aware write buffer. When it receives a write request, it checks the `inode` structure of the write request. We added a variable `(i_nrb)` to the `inode` structure that indicates whether the file is mapped to a certain fsync-aware write buffer. If the write request is for a fsynced file, it is inserted into a dedicated fsync-aware write buffer. Otherwise, the write request is inserted into a normal write buffer. This ensures that fsync-related data blocks exist only in a dedicated fsync-aware write buffer. Fig. 1 shows the buffers that should be flushed during fsync in MT-FTL and FA-FTL. For MT-FTL, the buffer as many as the number of cores should be flushed, but for FA-FTL, only one fsync-aware write buffer should be flushed.

In addition, the iJournaling technique [4] is used to ensure that journaling data is inserted into the same write buffer as normal data. In the compound transaction journaling of EXT4, it is impossible to distinguish the write buffer according to the `inode`, so all write buffers should be flushed. Here, fsync-aware write buffers are arbitrarily generated as many as the

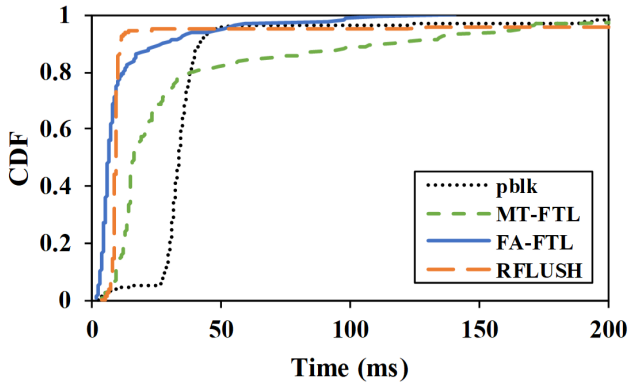
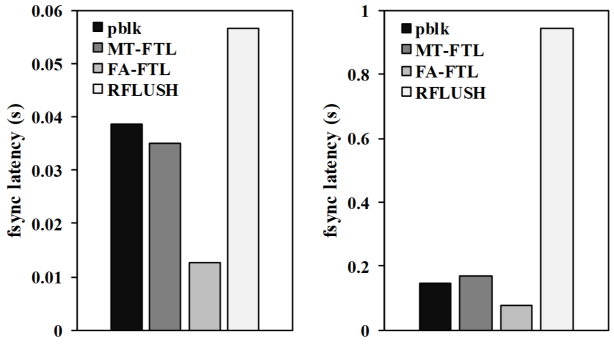


Fig. 2: CDF of fsync latencies



(a) Average latency (b) 95% tail latency

Fig. 3: Comparison of fsync latency

number of cores. A study on the proper number of fsync-aware write buffers according to the number of cores and workloads, and a technique for efficiently sharing a limited number of fsync-aware write buffers with fsynced files will be left as future works.

B. Alternative Design

Dedicated normal write buffer for fsynced file: This technique uses normal write buffer for buffering fsync-related data blocks. Instead, fsync-related data blocks are inserted only one normal write buffer. So it does not require additional memory. However, the dedicated normal write buffer to which the fsynced file is mapped includes not only the fsync-related data blocks, but also the data blocks for the other file and the asynchronous data blocks.

Fsync-aware write buffer per fsynced file: This technique dynamically allocates the fsync-aware write buffer per fsynced file. Although this technique consumes more memory than other techniques, but fsync latency is optimal because it only flushes fsync-related data blocks during fsync. However, the cost of dynamic allocation and deallocation has a negative impact on system performance.

III. Evaluation

A. Experimental Setup

We emulated the Guest OS and OC-SSD using the qemu-nvme emulator [2]. Guest OS emulation environment is as follows: 4 cores, 16GB DRAM, Ubuntu 16.04.3, Linux kernel 4.13.0-rc2. The OC-SSD emulation environment is as follows: 16 LUNs, 4 planes, 4GB block, 16KB page, 4KB sector.

All experiments were performed with iJournaling. The application scenarios used are as follows: Four threads write 2GB each, and four threads write 400KB each and then call the fsync system call.

B. Experimental Results

Fig. 2 shows a CDF representation of the fsync latencies of pblk, MT-FTL, FA-FTL, and RFLUSH [5]. MT-FTL has improved fsync latencies over pblk due to multiple write buffer and writer thread. FA-FTL flushes only one fsync-aware write buffer during fsync, resulting in fewer flush commands than MT-FTL. Fig. 3a shows that FA-FTL improves the fsync latency by 141% over pblk and 119% over MT-FTL. In addition, RFLUSH also improved fsync latency by searching and flushing only fsync-related data blocks during fsync. However, Fig. 3b shows that the 95% tail latency of RFLUSH is about 12 times that of FA-FTL. In the RFLUSH, fsync-related data blocks may exist in different write buffers. Therefore, more flush commands may occur than FA-FTL because the write request in different write buffers does not include in a single flush command.

IV. Conclusion

The proposed host FTL (FA-FTL) uses fsync-dedicated write buffer to improve the fsync latency over existing host FTLs. In existing host FTLs, all write buffers should be flushed to storage during fsync. FA-FTL flushes only fsync-aware write buffer mapped to fsynced file during fsync. Therefore, fsync latencies are improved because it flushes less data blocks which not related to fsync.

Acknowledgment

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2016R1A2B2008672)

References

[1] Bjørling, Matias, Javier González, and Philippe Bonnet. ”LightNVM: The Linux Open-Channel SSD Subsystem.” 15th USENIX Conference on File and Storage Technologies (FAST 17). 2017.

[2] OpenChannelSSD. “qemu-nvme,” https://github.com/OpenChannelSSD/qemu-nvme. 2019.

[3] Jhin, Jhuyeong, Hyukjoong Kim, and Dongkun Shin. ”Optimizing Host-level Flash Translation Layer with Considering Storage Stack of Host Systems.” Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication. ACM, 2018.

[4] Park, Daejun, and Dongkun Shin. ”iJournaling: Fine-grained journaling for improving the latency of fsync system call.” 2017 USENIX Annual Technical Conference (USENIXATC 17). 2017.

[5] Yeon, Jeseong, et al. ”RFLUSH: Rethink the Flush.” 16th USENIX Conference on File and Storage Technologies (FAST 18). 2018.

[6] H. Qin, D. Feng, W. Tong, J. Liu and Y. Zhao, ”QBLK: Towards Fully Exploiting the Parallelism of Open-Channel SSDs,” 2019 Design, Automation Test in Europe Conference Exhibition (DATE), Florence, Italy, 2019, pp. 1064-1069.