WOJ: Enabling Write-Once Full-data Journaling in SSDs by Using Weak-Hashing-based Deduplication

Fan Ni
University of Texas at Arlington
Arlington, Texas, US
fan.ni@mavs.uta.edu

Xingbo Wu
University of Illinois at Chicago
Chicago, Illinois, US
wuxb@uic.edu

Weijun Li
Shenzhen Dapu
Microelectronics Co. Ltd
Shenzhen, China
liweijun@dputech.com

Lei Wang
Beihang University
Beijing, China
wanglei@buaa.edu.cn

Song Jiang
University of Texas at Arlington
Arlington, Texas, US
song.jiang@uta.edu

Keywords

Journaling, File systems, Deduplication, SSD

1. INTRODUCTION

Journaling is a commonly used technique to ensure data consistency in file systems, such as ext3 and ext4. With journaling technique, file system updates are first recorded in a journal (in the *commit* phase) and later applied to their home locations in the file system (in the *checkpoint* phase). Based on the contents recorded in the journal, file sytem can be either in data or metadata journaling mode. With data journaling mode enabled, all file system (data and metadata) updates are written to the journal before being written to the files later on. In contrast, with metadata journaling mode, only updated metadata are written to and protected by the journal, while data are written directly to their home locations in the files. File system users are usually reluctant to use the data journaling mode as every modification (data and metadata) to the file system is written twice, and instead resort to metadata journaling for its fast speed.

However, metadata journaling has several limitations. First, updated data blocks can be mixed with un-updated ones in any order even with sequential write pattern if the system crashes during the file's overwriting. Second, it cannot guarantee even the consistency of metadata. For example, with the metadata journaling in the *ordered* mode in ext3 and ext4, the modify time ("mtime") of a file may remain unchanged after the file is updated. This metadata inconsistency can raise an issue with applications relying on the *mtime* attribute to decide their next actions, including GNU make and file integrity checks using signatures. Last but not least, recent research has revealed that metadata journaling is prone to introduce vulnerabilities to user-level applications as it reorders applications' write operations for performance [1]. To avoid the vulnerabilities, application developers have to be aware of write re-ordering in file systems and eliminate their side effects with extra efforts in their programs, such as inserting extra flushes between file system operations to enforce the right order. However, it is not easy to avoid the vulnerabilities at the application level even for expert programmers. These vulnerabilities are disclosed in widely-used applications [2], including LevelDB and Git.

As Linus Torvalds stated "*Filesystem people should aim to make 'badly written' code 'just work*" [3], use of data journaling adheres to the belief. With data journaling, file system maintains a global write order and preserves application order for both metadata and data, which provides the strongest data consistency support for applications. Most crash-consistency vulnerabilities can be avoided with data journaling, and the remaining ones have minor consequences and are readily to be masked or fixed [1, 2]. With the clear advantages in providing data reliability and stronger file system consistency support for applications, data journaling becomes increasingly necessary. With the wide use of fast SSDs, data journaling's higher demand on write bandwidth is likely to be accommodated and the once-thought expensive journaling approach may become affordable.

While SSD can provide much higher write throughput to potentially support data journaling well, its endurance becomes a new barrier to adoption of data journaling, which doubles write traffic to the disk. For most flash-based SSDs, each cell can only be written several thousand times in its lifespan. In this way, addressing the write-twice issue is a must for use of data journaling on SSDs. With data journaling, the modifications made to the file systems are first written to the journal and then to the files. Since the data written to the files are identical to those to the journal, it is possible to apply deduplication technique to avoid writing the data twice. However, existing deduplication techniques can be too expensive for fast SSDs in terms of computation, space, and synchronization overheads as they all rely on collision-resistant fingerprints, or hash values computed on data contents, to identify duplicate data. Example hash functions include SHA-1 and MD5. The computation overhead can be substantial as each block write requires computing its fingerprint. According to our measurements, the time spent on computing the fingerprint of a block can be longer than writing the block to an NVMe SSD device, indicating fingerprint computation becomes a new performance bottleneck on the I/O path. While computing weaker fingerprints, such as CRC32 values, can be tens or hundreds of times faster, it can compromise correctness. In addition

IFIP WG 7.3 Performance 2018 Toulouse, France

to the computation cost, existing deduplication technique consumes plenty of memory space for caching its metadata. And it requires periodical synchronization of the metadata onto the disk on a timely manner for data reliability and short response time. Moreover, different types of metadata should be persisted onto the disk in a particular order. All these require frequent use of expensive *flush* operations.

In this work we propose a solution that is built in SSDs to transparently support **W**rite-**O**nce data **J**ournaling, named WOJ. While it still uses block-deduplication to remove the second writes, it addresses all three issues with the regular deduplication technique. First, WOJ can use ultra-lightweight non-collision-resistant hashing to identify second writes of duplicated blocks without compromising correctness. Second, WOJ only maintains a small amount of metadata, which is thousand times smaller than that of regular deduplication and can be fit in the SSD's internal memory. Third, WOJ integrates its block mapping with SSD's FTL and does not require frequent flushes from the host to the device. Our experiments with a wide range of micro and real-world benchmarks show that WOJ removes about half of the writes in data journaling and provides significant performance improvement over existing deduplication schemes.

2. DESIGN OF WOJ

The design of WOJ addresses two main challenges: (1) use of ultra-lightweight fingerprints without compromising correctness, and (2) caching very small amount of metadata whose size is decoupled from the SSD's capacity.

WOJ uses non-collision-resistant fingerprints to detect duplicate blocks in the second writes in data journaling with very low overheads. The key difference in its use of fingerprints from regular deduplication is that WOJ does not need to determine the existence of a duplicate block in the second write (in checkpoint phase) as it is guaranteed by the journaling operation, which is *"all new data is written to the journal first, and then to its final location"* [4]. As long as a fingerprint is not shared by more than one block in the journal, it can be used to identify the corresponding block in the checkpoint phase (even if non-collision-resistant fingerprints are used) and avoid writing it to the disk.

To facilitate the identification, WOJ maintains a fingerprint pool. A fingerprint is inserted in the pool and used for detecting duplicate blocks in the checkpoint phase only when two conditions are satisfied. First, the fingerprint is computed over the contents of a block in the commit phase. Second, the fingerprint is unique. For each fingerprint in the pool, it is associated with a unique physical page address (PPA) where the corresponding block of data is stored. In the commit phase, When a block is written into the journal, its fingerprint is computed (1.1 in Figure 1) and the pool is searched for the fingerprint. If not found, it is added into the pool (1.3 in Figure 1). Otherwise, a collision occurs (1.2 in Figure 1). To avoid mis-deduplication, the fingerprint is marked as invalid and deduplication attempt on blocks with the fingerprint is aborted. The fingerprint pool can be very small, whose size is capped by number of blocks in a journal. Note that a journal is usually configured as a few hundreds of megabytes to several gigabytes. A fingerprint is removed from the pool right after it is used for a successful removal of a second write. Otherwise, it will be removed when the space for its corresponding block is reclaimed in the journal. In either case the lifetime of a fingerprint in the pool

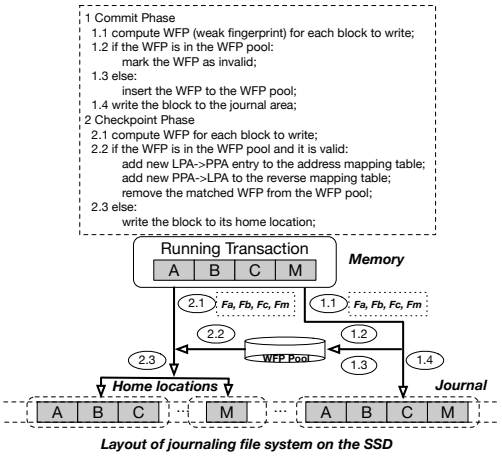

Figure 1: ***Operations in WOJ for committing and checkpointing blocks.*** F_a, F_b, F_c, ***and*** F_m ***are weak fingerprints of blocks*** A, B, C, ***and*** M, ***respectively.***

is short and the collision is expected to be rare. Note that WOJ does not de-duplicate blocks within a journal.

In the checkpoint phase, for each writing block, its fingerprint is computed and compared to those in the pool (2.1 in Figure 1). If it matches a valid fingerprint (one that has not experienced any collision), the block write is deduplicated (2.2 in Figure 1). Only an entry in the FTL's address mapping table (from a block's logical page address (LPA) to its logical page address (PPA)) is updated to reflect that the block's LPA is mapped to PPA of the block with the matching fingerprint. If the block matches an invalid fingerprint (2.3 in Figure 1), it is written to the disk as usual.

The design of WOJ also considers other aspects for its efficient use in an SSD, including mechanism of passing file-system-level knowledge to SSD to enable WOJ and data structures for efficient garbage collection and wear leveling.

3. ACKNOWLEDGEMENTS

This work was mainly supported by US National Science Foundation under CNS 1527076. In addition, Weijun Li was supported by Shenzhen Peacock Plan (KQTD20150917164 53118), and Lei Wang was supported by National Natural Science Foundation of China (No. 61672073).

4. REFERENCES

[1] T. S. Pillai, R. Alagappan, L. Lu, V. Chidambaram, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, Application Crash Consistency and Performance with CCFS, in: FAST'17, USENIX Association, Berkeley, CA, USA, 2017, pp. 181–196.

[2] T. S. Pillai, V. Chidambaram, R. Alagappan, S. Al-Kiswany, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, All file systems are not created equal: On the complexity of crafting crash-consistent applications, in: OSDI'14, USENIX Association, Berkeley, CA, USA, 2014, pp. 433–448.

[3] L. Torvalds, Linux 2.6.29., `https://lwn.net/Articles/326505/` (2009).

[4] L. Kernel, Ext4 Filesystem, `https://www.kernel.org/doc/Documentation/filesystems/ext4.txt` (2017).