Reducing Excessive Journaling Overhead in Mobile Devices with Small-Sized NVRAM

Junghoon Kim[†], Changwoo Min[†‡], and Young Ik Eom[†]
[†]Sungkyunkwan University, Korea [‡]Samsung Electronics, Korea
{myhuni20,multics69,yieom}@skku.edu

***Abstract*—The excessive journaling degrades the performance and shortens the lifetime of NAND flash storage in mobile devices. We propose a novel journaling scheme that resolves this problem by using small-sized NVRAM efficiently. Experimental results show that our proposed scheme outperforms EXT4 by up to 16.8 times for synthetic workloads. Also, for TPC-C SQLite benchmark, it enhances the transaction throughput by 20% and reduces the number of journal writes by 58% with only 16 MB NVRAM.**

I. Introduction

Reliability is one of the most important issues for designing file systems in battery-powered mobile devices, such as smartphones and tablets, because sudden power failures are more likely to occur. Journaling is a standard technique for restoring a file system to a consistent state in mobile devices. In the journaling file system, updated data are first written to a journal area for system recovery and then periodically written to their home locations, i.e., original locations. However, these duplicated writes impose two serious problems especially in mobile devices, where NAND flash storage is used: degrading the performance and shortening the lifetime of NAND flash storage, which has limited P/E cycles [1]. A recent study [2] shows that the excessive journaling is the major source of the performance degradation in smartphones. A state-of-the-art solution to reduce the journaling overhead is to use non-volatile memory (NVRAM), such as PCM or STT-MRAM, as the union of buffer cache and journal area [1]. However, it is difficult to apply this solution in mobile devices due to limited density and high cost of NVRAM [3].

In this paper, we propose a novel journaling scheme, called *Delta Journaling (DJ)*, to reduce the journaling overhead in NAND flash storage by using *small-sized NVRAM*. DJ exploits the byte-addressable and non-volatile characteristics of NVRAM. Before writing an updated block to the journal area, we first calculate the difference, or delta, between the updated and original block. If the compression ratio of the delta is high, only the compressed delta is written to the delta journal area in the NVRAM. Otherwise, the non-compressed block is written to the journal area in NAND flash storage. This approach is based on the unique update pattern in mobile devices: *most updates are very small.*

Previous study shows that SQLite database operations generate approximately 80% of writes in mobile devices [2]. SQLite frequently calls `fsync` system call to guarantee data consistency. The frequent `fsync` calls incur the excessive journaling and thus seriously injure the overall system performance [2], [4]. Figure 1 shows the cumulative distribution of the difference between updated and original block while running TPC-C benchmark with SQLite. More than 70% of updated blocks have less than 10% difference (i.e., the difference size is smaller than 400 bytes, compared with original block). This observation implies that our approach, which uses the small-sized NVRAM as the compressed delta journal area, can greatly improve the performance and the lifetime of NAND flash storage in mobile devices.

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Plannig(2010-0020730).

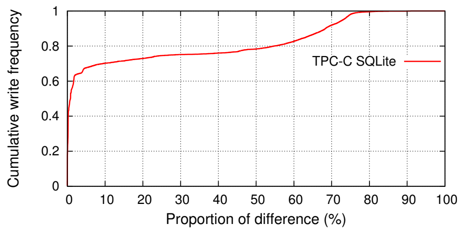


Fig. 1: The cumulative distribution of the difference in updating blocks

It is also possible to make the compressed delta in the NAND flash storage layer [3], [5]. In this case, computing cost and power consumption are high because all updated blocks must be taken into account for the delta compression without knowing the file system level semantics. Also, it is ineffective to use a less powerful internal micro-processor of NAND flash storage [5] or require additional hardware for the delta compression [3]. On the other hand, DJ only considers overwrite blocks for the delta compression by exploiting file system level semantics and also it can reduce the compression time by using a more powerful host processor.

II. Design of Delta Journaling

Figure 2 shows the overview of DJ. Same to the original journaling technique, DJ also performs *commit* and *checkpoint* operations: the commit operation writes the updated data to the journal area and then the checkpoint operation periodically writes the updated data to their home locations. Our DJ is incorporated into the commit operation. DJ first calculates how much the updated block is different from the original block in bit-wise by using XOR operation. If the result of XOR is dominated by 0 or 1 (i.e., the compression ratio is expected to be high), DJ logs the compressed delta on the delta journal area of NVRAM. Otherwise, the updated block is stored in the journal area of NAND flash storage. More detail design of DJ is as follows.

A. Difference Capturing

Whenever the commit operation is triggered, DJ captures the difference between the updated and original block. To do this, DJ maintains original blocks in DRAM memory to avoid

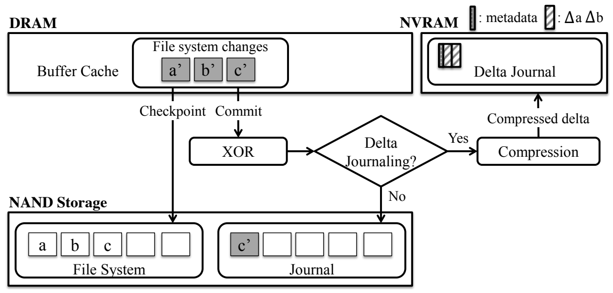


Fig. 2: The overview of Delta Journaling

reading them from NAND flash storage. When data is modified by a process, the operating system fetches original blocks into the buffer cache and DJ copies original blocks before making modification. After the updated block is committed, its copied original block is also dropped by DJ. Through this process, DJ can efficiently capture the difference without additional IO and large memory overhead.

B. *Delta Compression*

If the difference meets certain criteria, the differential data block, made from the XOR operation, is compressed and then stored in the NVRAM. DJ uses LZO lossless compression algorithm, which makes a good trade-off between compression time and compression ratio among several candidates such as fastLZ, gzip, and LZ4. In our experimental results, if the difference is either less than 25%, or more than 75%, the compression ratio is high (i.e., compressed delta size is expected to be less than 25% of the block size). Thus, we set the difference threshold of DJ prototype as above.

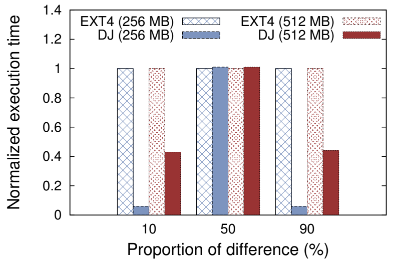
C. *NVRAM Management*

Checkpoint operation is triggered when the time interval has passed after the last checkpoint or the free space ratio of the journal area becomes below the threshold. At this point, DJ updates the file system with committed data. If a system crash occurs during the checkpoint operation, DJ restores the file system to a consistent state by reflecting journaled data, stored in NAND flash storage or NVRAM, into their home locations. In case of a delta stored in NVRAM, the journal block is calculated by XOR operation between its original block in NAND flash storage and decompressed delta in NVRAM. To do this, DJ manages metadata in the NVRAM including journal transaction number, home location, offset, and length of delta.

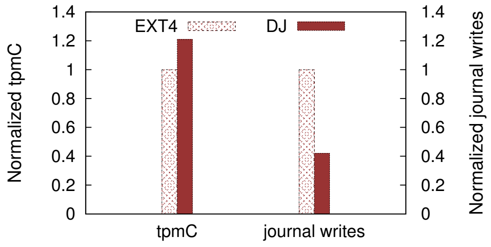
III. Performance Evaluation

We implemented DJ on Linux kernel 3.4 and evaluated its performance on a mobile device equipped with a dual-core mobile CPU and 2GB DRAM. For NAND flash storage, 16 GB A-Data microSD card is used. Since there is no commercially available NVRAM for mobile devices, 16 MB of the DRAM is used for small-sized NVRAM region [1]. As a baseline for comparison, EXT4 file system is mounted with journal mode, which logs both data and metadata to provide the same consistency level as DJ.

To evaluate the best and worst case performance of DJ, we developed synthetic workloads, which consist of 256 MB and 512 MB overwrites with various proportions of difference. As shown in Figure 3a, in case of 10% and 90% difference, DJ outperforms EXT4 by 16.8 and 2.3 times in 256 MB and 512 MB overwrites, respectively. The performance improvement of



(a) Synthetic workload



(b) TPC-C SQLite

Fig. 3: Comparison between DJ and EXT4

256 MB overwrites is far higher because DJ does not trigger checkpoint operation caused by the lack of the journal area. This result indicates that DJ can delay checkpoint operation by using small-sized NVRAM efficiently. In case of 50% difference, which is our worst case scenario with no writing to the delta journal area, the performance of DJ is approximately same to that of EXT4. This result indicates that there is very little computing overhead for capturing difference.

For more realistic workload on mobile devices, we run TPC-C SQLite benchmark. Figure 3b shows the transaction processed per minute (tpmC) and the number of journal writes in the microSD card. DJ achieved about 20% improvement in transaction throughput than EXT4. Also, the number of journal writes in microSD card is reduced by 58%. This is because most small updated blocks are stored as the compressed deltas in the NVRAM. Through this, DJ can improve the performance and the lifetime of NAND flash storage. We also identify that average delta size per block is only 3.7% of the block size, about 150 bytes. This result indicates that DJ only compresses the differential data block in case of high compression ratio for efficient use of NVRAM.

IV. Conclusion

We proposed a novel journaling scheme for reducing the excessive journaling overhead in mobile devices. The proposed scheme stores a commit block as a compressed delta in the small-sized NVRAM only when the compressed delta is small enough. Experimental results show that the proposed scheme can improve the performance and the lifetime of NAND flash storage with only 16 MB NVRAM.

References

[1] E. Lee, H. Bahn, and S. H. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," in *Proc. of FAST*, 2013.
[2] K. Lee and Y. Won, "Smart layers and dumb result: IO characterization of an android-based smartphone," in *Proc. of EMSOFT*, 2012.
[3] S. Lee, S. Jung, and Y. H. Song, "An efficient use of PRAM for an enhancement in the performance and durability of NAND storage systems," *IEEE Transactions on Consumer Electronics*, 2012.
[4] W.-H. Kang, S.-W. Lee, B. Moon, G.-H. Oh, and C. Min, "X-FTL: Transactional FTL for SQLite databases," in *Proc. of SIGMOD*, 2013.
[5] G. Wu and X. He, "Delta-FTL: Improving SSD lifetime via exploiting content locality," in *Proc. of EuroSys*, 2012.