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

Junghoon Kim, Changwoo Min, and Young Ik Eom

Abstract — *Journaling techniques are widely used to guarantee file system consistency of battery-powered mobile devices such as smartphones and tablets. In a journaling file system, system recovery is facilitated by first writing updated data blocks to a journal area and then periodically writing them to their home locations. However, these duplicated writes degrade the performance and shorten the lifetime of NAND flash storage in mobile devices. In particular, a lightweight database library, which is mainly used to manage application data in mobile devices, is a major cause of excessive journaling because it frequently triggers the costly file synchronization to guarantee the atomicity of transactional execution and thus generates a significant amount of synchronous random write traffic. This paper presents a novel journaling scheme, called Delta Journaling (DJ), to resolve this problem efficiently by using small-sized nonvolatile random access memory (NVRAM). DJ is based on a unique update pattern found in mobile devices, where file system updates are mostly very small. By exploiting the byte-addressable and the nonvolatile characteristics of NVRAM, DJ stores a journal block as a compressed delta in the small-sized NVRAM only when the compressed delta is small enough. Experimental results show that DJ outperforms a traditional journaling file system by up to 16.8 times for synthetic workloads. For a real-world workload, it enhances transaction throughput by 25.5% and 29.2% in ordered and journal modes, respectively, with only 16 MB NVRAM. Also, DJ enhances the lifetime of NAND flash storage by eliminating almost all journal writes without any loss of reliability.*[1]

Index Terms — Journaling file system, NAND flash storage, NVRAM, Reliability

I. INTRODUCTION

File system reliability is one of the most crucial concerns when designing file systems for battery-powered mobile devices, such as smartphones and tablets, because sudden power failure is more likely to occur. Journaling is a standard technique used in mobile device to restore a file system to a consistent state [1], [2]. In a journaling file system, system recovery is facilitated by first writing updated data blocks to a journal area and then periodically writing them to their home locations (i.e., their original locations). However, these duplicated writes degrade the performance and shorten the lifetime of NAND flash storage, especially in mobile devices where NAND flash memory is used for main data storage as a result of several limitations of NAND flash hardware [3], [4]. First, once a page is programmed, the page must be erased in order to write new data in the same place. Moreover, an erase operation takes about 10 times longer than a write operation because NAND flash memory can be erased only in the unit of a block, which is much larger than a page [5], [6]. Thus, this erase-before-write constraint makes the cost of a write operation that is much higher than that of a read operation. Second, NAND flash memory has limited program/erase cycles. Usually, multi-level cell (MLC) NAND flash memory guarantees about 10,000 erase cycles for each block, and triple-level cell (TLC) NAND flash memory guarantees only about 1,000 erase cycles [5].

In order to reduce the journaling overhead, previous studies have proposed various solutions [7]-[10]. However, these solutions still have limitations. Choi *et al.* [7] proposed a journal remapping technique that remaps addresses of the logged journal data to addresses of the home locations instead of writing the updated blocks once more to the NAND flash storage. However, when writing data or metadata to NAND flash storage upon a periodic flush or upon a synchronization request such as an *fsync* system call, the updated blocks must be written to a separate journal area synchronously, regardless of the size of updates. Lee *et al.* [8] proposed a buffer cache architecture that uses nonvolatile random access memory (NVRAM), such as phase-change memory (PCM) or spin-transfer torque magnetic RAM (STT-MRAM), as the union of buffer cache and journal area. However, it is difficult to implement this solution for mobile devices due to the limited density of NVRAM [11]. Moreover, the installation of larger NVRAM chips in mobile devices, such as smartphones and tablets, entails higher material costs, and so it deteriorates the competitiveness of the products in the markets. Lee *et al.* [9] proposed a journaling file system that reduces the amount of writes produced during journaling by exploiting the byte-

[1] This work was supported by the IT R&D program of MKE/KEIT [10041244, Smart TV 2.0 Software Platform]. This research was supported by the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-4006) supervised by the NIPA (National IT Industry Promotion Agency).

Junghoon Kim and Young Ik Eom are with the College of Information and Communication Engineering, Sungkyunkwan University, 2066, Seobu-ro, Jangan-gu, Suwon 440-746, South Korea. (e-mail: myhuni20@skku.edu, yieom@skku.edu).

Changwoo Min is with the Software Center, Samsung Electronics, 129, Samsung-ro, Yeongtong-gu, Suwon 443-742, South Korea. (e-mail: changwoo.min@samsung.com).

Contributed Paper
Manuscript received 03/27/14
Current version published 06/23/14
Electronic version published 06/23/14.

accessibility of PCM. However, this solution can only be applied to a system that specifically uses PCM as the main data storage. Kim *et al.* [10] proposed a journaling technique for a lightweight database library. However, this solution can only be applied to a database library that is specifically implemented in the rollback journal mode, instead of the write-ahead log mode.

This paper proposes a novel journaling scheme, called *Delta Journaling (DJ)*, to reduce journaling overhead in mobile devices by using *small-sized NVRAM* efficiently. DJ uses two separate journal areas, one in the NAND flash storage and another in the NVRAM, and it exploits the byte-addressable and the nonvolatile characteristics of NVRAM. Before writing an updated block to the journal area, DJ first calculates the difference (i.e., the delta) between the updated and the original blocks. If the compression ratio of the delta is high, only the compressed delta is logged in the delta journal area of the NVRAM. Otherwise, the updated block is written to the journal area of the NAND flash storage. Alternatively, it is also possible to write the compressed delta to the NAND flash storage layer [11], [12]. In this case, computing cost and power consumption are high because all updated blocks must be taken into account when compressing the delta without knowing the file system level semantics. Also, it is ineffective to implement additional hardware for the delta compression [11] or to use a less powerful internal microprocessor in the NAND flash storage [12]. 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. DJ is especially efficient in mobile devices, where file system updates are mostly very small.

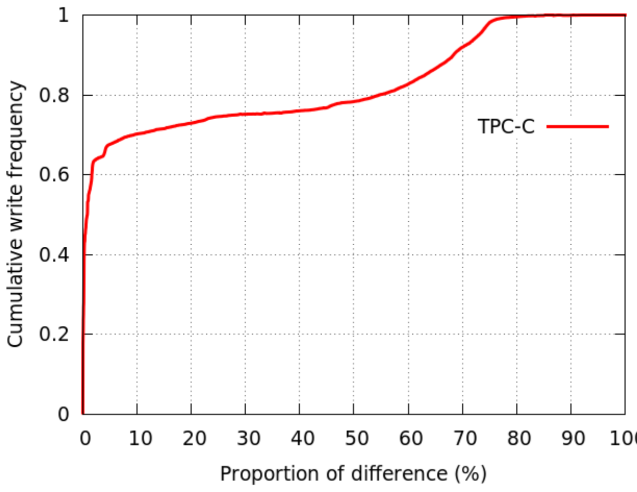

Fig. 1. Cumulative distribution of the differences between updated and original blocks.

A previous study showed that a lightweight database library, which is mainly used for managing mobile application data, generates approximately 80% of the writes in a mobile device [3]. The lightweight database operation frequently triggers the *fsync* system call to guarantee data consistency. The frequent *fsync* calls incur excessive journaling and thus negatively impact overall system performance [13], [14]. Fig. 1 shows the cumulative distribution of the differences between updated and original blocks while running the TPC-C benchmark with a lightweight database library. More than 70% of updated blocks have less than 10% difference (i.e., the size of the difference when compared to the original block is smaller than 400 bytes). This observation implies that DJ, which uses small-sized NVRAM as the journal area for the compressed delta, can greatly improve the performance and the lifetime of NAND flash storage in mobile devices.

The remainder of this paper is organized as follows: Section II presents related work, Section III describes the detailed design of DJ, Section IV presents the experimental results, and finally, Section V concludes the paper.

II. Related Work

DJ can greatly reduce the journaling overhead in mobile devices by exploiting the byte-addressable and the nonvolatile characteristics of NVRAM. In this regard, the two types of existing technologies that are most closely related to DJ are the NVRAM/NAND flash hybrid architectures [5], [11], [15]-[19] and techniques for reducing the journaling overhead [7]-[10]. This section briefly presents each work and compares it to DJ.

A. NVRAM/NAND Flash Hybrid Architecture

The NVRAM/NAND flash hybrid architecture utilizes the byte-accessibility and in-place update characteristics of NVRAM to complement the hardware limitations of NAND flash storage. The first approach employs a log region in PCM, storing the updates of data pages in the form of logs [15]. The log region allows in-place updates so that frequent, small updates can be effectively absorbed. However, when large updates are performed, this approach increases write latency because the read/write latency of PCM is slower than that of NAND flash memory for large data [20]. In contrast, DJ stores a compressed delta in the NVRAM only when the compressed delta is small enough. The second approach employs NVRAM as a metadata storage component [5], [16]-[19]. In general, file system metadata is frequently updated and only a few bytes are actually modified. Thus, in order to reduce excessive garbage in NAND flash memory, the metadata storage component stores file system metadata in PCM [5], [16]-[18] or synchronously stores the metadata updates as logs in NVRAM [19]. In contrast, DJ considers both file system data and metadata for the delta compression by exploiting a small update pattern in mobile devices. The third approach uses PCM as a buffer space in a storage device [11]. However, this approach requires additional hardware for the delta compression and cannot be applied to traditional log-structured file systems [21], [22] because it does not know file system level semantics. In contrast, DJ uses a more powerful host processor and exploits file system level semantics for the delta compression.

B. Technique for Reducing Journaling Overhead

Modern file systems in mobile devices use journaling to guarantee the consistency of both file system data and metadata [1], [2]. In journaling file systems, file system updates are first recorded in a separate journal area before being written back to their home locations in case the system needs to perform a recovery. However, journaling has considerable overhead because the same file system changes are written to the storage device twice.

Choi *et al.* [7] proposed a journal remapping technique that remaps addresses of the logged file system changes to addresses of the home locations of the changes. With this technique, the journal pages are altered into updated pages of the file system, so additional writing to NAND flash storage is prohibited. However, when writing the updates to NAND flash storage upon periodic flush or upon synchronization request, the updated blocks must be written to a separate journal area of the NAND flash storage synchronously, regardless of the size of updates. Lee *et al.* [8] proposed a buffer cache architecture that subsumes the function of caching and journaling in a unified NVRAM space. Blocks in the buffer cache are converted to journal logs by using *in-place commit* technique, so the frequency of storage accesses can be reduced. However, it is difficult to apply this solution in mobile devices due to limited density and high cost of NVRAM [11]. In contrast, DJ stores journal data as a compressed delta in the NVRAM when a high compression ratio is achieved. Thus, DJ can greatly reduce the synchronous writes of journaling in mobile devices with only a small-sized NVRAM.

Lee *et al.* [9] proposed a journaling file system that reduces the amount of journal writes considering the PCM characteristics. However, that system is designed for devices where NVRAM is used for main data storage. In contrast, DJ focuses on use in mobile devices, where NAND flash memory is used for main data storage. Kim *et al.* [10] proposed a journaling technique for a lightweight database library. Before reflecting new data to the original location of the database files, the lightweight database library creates a rollback journal file that maintains the original data for system recovery. To improve storage performance, the proposed technique places the rollback journal file in NVRAM. In contrast, DJ can be applied to all files, including database files, because it is designed at the file system level.

The previous version of DJ [23] stores journal data as a compressed delta in a small-sized NVRAM only when the compressed delta is small enough. Through this, DJ can reduce the excessive journaling overhead in mobile devices. Although it utilizes a small-sized NVRAM as a delta journal area efficiently, it does not consider the optimal checkpoint threshold of NVRAM, and thus, it cannot fully utilize the NVRAM space. Also, there are more opportunities to optimize the journaling performance. In this regard, the current version of DJ finds out the optimal checkpoint threshold of NVRAM and extends targets to include the journal descriptor (JD)/journal commit (JC) block to provide better performance.

III. Design of Delta Journaling

This section first describes the overall architecture of the proposed system, and then, the detailed design of DJ is described in the following sequence: difference capturing, delta compression, NVRAM management, and system recovery.

A. System Overview

Fig. 2 shows the overall architecture of the proposed system. In the proposed system, the buffer cache acts as a transparent cache for storage-backed pages, and it is kept in the DRAM for fast access. To reduce the journaling overhead, a small-sized NVRAM is connected to a memory bus supporting byte-level access. NAND flash storage is used as main data storage in the proposed system.

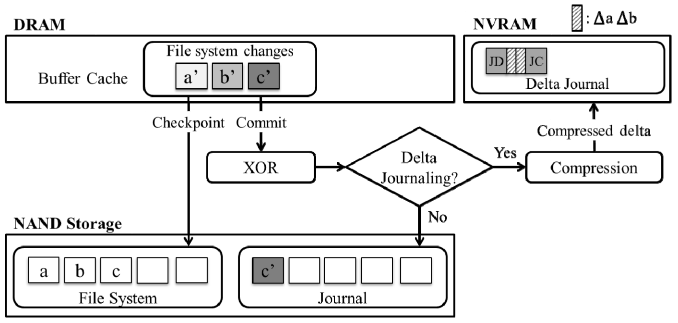

Fig. 2. Overall architecture of the proposed system.

Similarly to the original journaling technique, DJ also performs *commit* and *checkpoint* operations. The commit operation writes the updated data to the journal area for potential use during system recovery upon a periodic flush or upon a synchronization request, such as *fsync* system call, and the checkpoint operation updates the file system with the committed data when a specific time interval has passed after the last checkpoint or the free space ratio of the journal area crosses a predefined threshold. DJ is incorporated into the commit operation. Whenever a system call modifies the file system data, dirty data and metadata are grouped in the transaction and are handled in an atomic way. In the example, as shown in Fig. 2, a write request modifies the contents of blocks *a'*, *b'*, and *c'*. These dirty blocks are grouped in a transaction and are handled together. When a commit operation is triggered, DJ first writes a JD containing an array of journal block tags in the delta journal area of NVRAM. The array of journal block tags describes the home locations of the journal data in sequence. The details of the JD will be further discussed in Section III.D. Then, DJ calculates how much the updated block is different from the original block in bit-wise comparison by using the 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 in the delta journal area of NVRAM. In this example, DJ stores the blocks *a'* and *b'* as the compressed delta. Otherwise, the updated block is stored in the journal area of NAND flash storage as in the case of block *c'*. At the end of the commit, DJ writes a JC in the delta journal area of NVRAM to identify the end of the transaction. A more detailed design of DJ is as follows.

B. Difference Capturing

Whenever a commit operation is triggered, DJ captures the difference between the updated and original blocks as shown in Fig. 3. To do this, DJ maintains the original blocks in DRAM memory to avoid reading them from NAND flash storage. When the file system data is modified by a system call, the operating system fetches the original blocks into the buffer cache to minimize I/O operations. At this point, DJ copies the original blocks before making modification. Then, in the commit state, DJ calculates the difference level of each block by using the XOR operation. After an updated block is committed, the copy of the original block is also dropped by DJ. Through this process, DJ can efficiently capture the difference without incurring additional I/O operations and a large memory overhead.

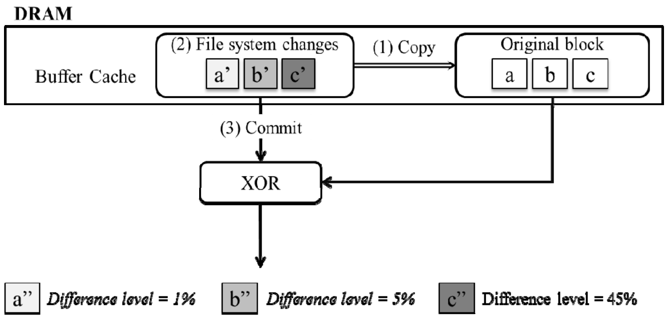

Fig. 3. Overall process of capturing the difference between the updated and original blocks.

C. Delta Compression

If the difference meets a certain criterion, the differential data block, made as a result of the XOR operation, is compressed and then stored in the journal area of NVRAM. For the delta compression, several candidate algorithms, including fastLZ, gzip, LZF, LZ4, LZO, and LZW, are investigated. All experiments were performed on a mobile device equipped with a dual-core mobile CPU (see Section IV for the detailed description of the environment). Fig. 4 and Fig. 5 show the compressed size and compression time of a 4 KB block among those lossless compression algorithms. The results showed that the gzip and LZW algorithms can achieve a high compression ratio (i.e., the compressed size is relatively smaller than that of the other candidate algorithms) as shown in Fig. 4. However, they also have high computing cost, as shown in Fig. 5.

Among the candidate algorithms, the LZO lossless compression algorithm, which has an adequate trade-off between compression time and compression ratio, is thus implemented in DJ. In the experimental results, if the difference was either less than 25% or more than 75%, the compression ratio was considered high (i.e., the compressed delta size is expected to be less than 25% of the block size). Thus, the difference threshold of DJ is configured as above, and DJ compresses the differential data block only when this criterion is satisfied.

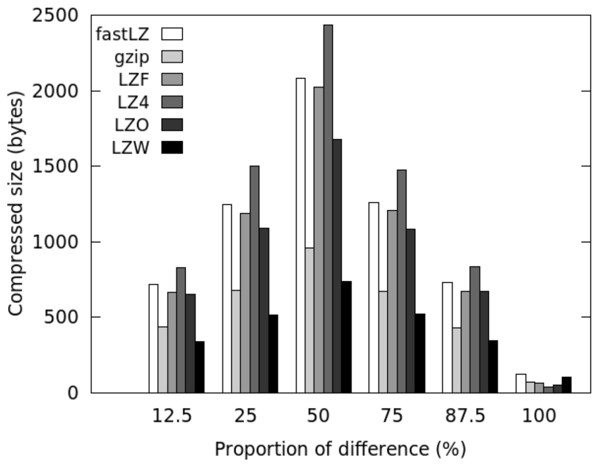

Fig. 4. Size of a 4 KB block compressed by several different lossless compression algorithms.

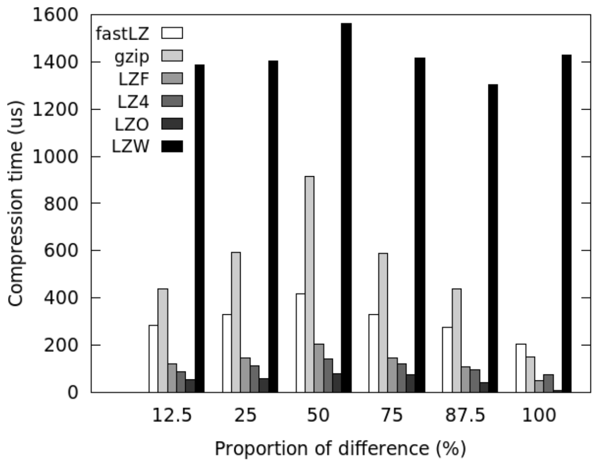

Fig. 5. Compression time of a 4 KB block by several different lossless compression algorithms.

D. NVRAM Management

DJ uses the NVRAM space as a delta journal area. Fig. 6 shows the layout of the NVRAM space and the data structure of JD and JC. A journal metadata contains the information of journal areas, one in the NAND flash storage and another in the NVRAM, including size of journal area, size of free space, and offset of the start position of each JD. During a commit operation, a JD is written to the delta journal area of the NVRAM first. The JD starts with a 12-byte journal header, which contains a magic header and a sequence number identifying the commit transaction, and the journal block tags follow the journal header. Each 16-byte journal block tag describes the home location of the journal data in sequence. There are two types of journal block tag, *TYPE_DELTA* and *TYPE_NON_DELTA*, to distinguish the journal blocks according to their locations. In the case of a journal block as a form of compressed delta in NVRAM, its journal block tag consists of the home location of the journal data, flags, offset, and length of the compressed delta. In the case of a journal block in NAND flash storage, its journal block tag consists of the home location of the journal data, flags, and the journal block number. The field for the flags is used to identify the tag type, and the field for the journal block number represents the block address of the journal data in the NAND flash storage. Through this process, DJ can maintain the journal areas in both NVRAM and NAND flash storage. At the end of the commit, a JC is attached to identify the end of the transaction. The JC also starts with a 12-byte journal header, and consists of a checksum and the commit time. The field for the checksum is used to improve reliability. The journal data in both NVRAM and NAND flash storage never plays an active role, until a system failure occurs, and can be recycled after a checkpoint operation is completed.

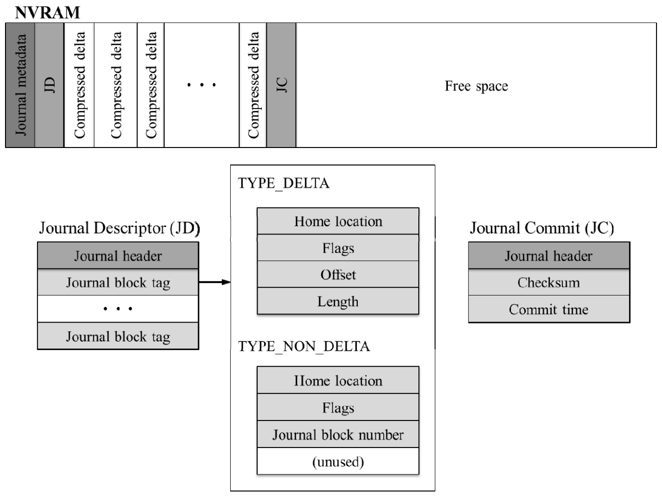

Fig. 6. Layout of the NVRAM and the data structure of JD and JC.

E. System Recovery

A sudden power failure is more likely to occur in battery-powered mobile devices, and it can result in an inconsistent state of the file system. After an abnormal termination, DJ restores the file system to a consistent state upon system reboot. There are two cases of a system crash to be considered. First, if a system crash occurs during a commit operation, the commit transaction may be partially logged in the journal space. In this case, DJ simply invalidates the journal records, which are described by the current commit transaction, in both NVRAM and NAND flash journal areas. Therefore, the file system remains as the last checkpoint state. Second, if a system crash occurs during a checkpoint operation, the checkpoint transaction may be partially reflected in the file system. At this point, the journal records play an active role. Since the partially reflected transaction still remains in the journal space, DJ restores the file system to a consistent state by reflecting the journal records, stored in NVRAM or NAND flash storage, into their home locations. In the case in which a compressed delta is stored in NVRAM, the journal record is calculated through an XOR operation between the original block in NAND flash storage and the decompressed delta in NVRAM. Through this process, DJ can recover the file system in a short time without performing time-consuming consistency checks on the whole file system.

IV. Performance Evaluation

First, this section presents the hardware and software configurations of the experimental platform. Then, this section describes the optimal checkpoint threshold of NVRAM. Finally, this section shows the experimental results of DJ and compares DJ to an ext4 file system.

A. Experimental Setup

DJ is implemented in the Linux kernel 3.4 and is evaluated on a real mobile device equipped with a dual-core CPU and 2 GB of DRAM. For the NAND flash storage, a 16 GB microSD card is used. Since there is no commercially available NVRAM for mobile devices, 16 MB of DRAM are allocated for use as a small-sized NVRAM region [8]. This is a reasonable assumption because the access time of NVRAM, such as PCM or STT-MRAM, is expected to be similar to that of DRAM [24], [25]. As a baseline for comparison, the ext4 file system is mounted with the *ordered* and *journal* modes. For performance reasons, only metadata blocks are logged in the ordered mode, and the ext4 file system uses the ordered mode as a default journal configuration. Unlike the ordered mode, the journal mode logs both data and metadata blocks, so the journal mode guarantees the highest data consistency. However, the journal mode has considerable overhead due to a large number of journal writes. According to the default configurations, the commit period is set to 5 seconds and the checkpoint operation is triggered when 5 minutes have passed after the last checkpoint or when a fourth of the journal area in the NAND flash storage is filled. The checkpoint threshold of the delta journal area is further discussed in Section IV.B.

B. Optimal Checkpoint Threshold of NVRAM

The previous version of DJ [23] did not consider the optimal checkpoint threshold of NVRAM, which was the same as that of the default configuration where a fourth of the delta journal area in NVRAM is filled. Thus, it cannot fully use the NVRAM space. For better space usage of NVRAM, the optimal checkpoint threshold of NVRAM is evaluated in this paper. For these experiments, three traces with different distributions are generated: the Zipfian distribution, Bathtub distribution, and Uniform distribution. In the Zipfian distribution, the difference between the updated and the original blocks is highly skewed in that most updates are very small. The Bathtub is a bimodal distribution where most updates are either very large or very small. Lastly, all difference sizes are evenly distributed in the Uniform distribution.

Fig. 7 shows the normalized execution time of each workload under different checkpoint thresholds of NVRAM. *W1*, *W2*, and *W3* represent the Zipfian, Bathtub, and Uniform distributions, respectively, and the execution time of each workload for DJ is normalized to that of ext4. The experimental results show that *W1* and *W2* achieve the best performance when the checkpoint threshold of NVRAM is set to 75%. On the other hand, *W3* has the same performance under all checkpoint thresholds.

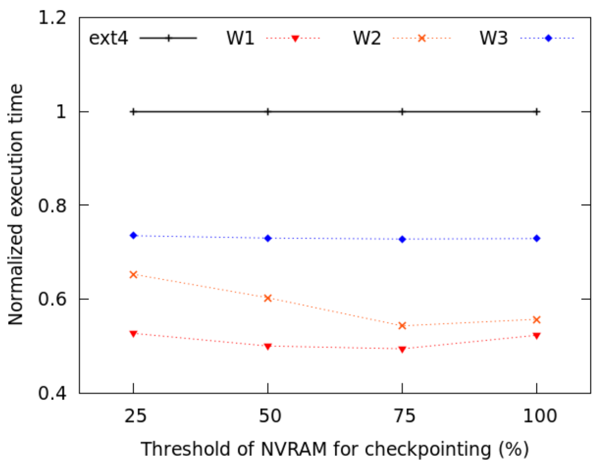

Fig. 7. Normalized execution time of each workload under different checkpoint thresholds of NVRAM.

The number of checkpoint operations was measured to analyze the experimental results. Fig. 8 shows the number of checkpoint operations of each workload under different checkpoint thresholds of NVRAM, and the results show that *W1* and *W2* have the smallest number of checkpoint operations when the checkpoint threshold of NVRAM is set to 75%. This result indicates that as the number of checkpoint operations decreases, the performance of DJ improves. *W3* has the same number of checkpoint operations under all checkpoint thresholds, unlike *W1* and *W2*.

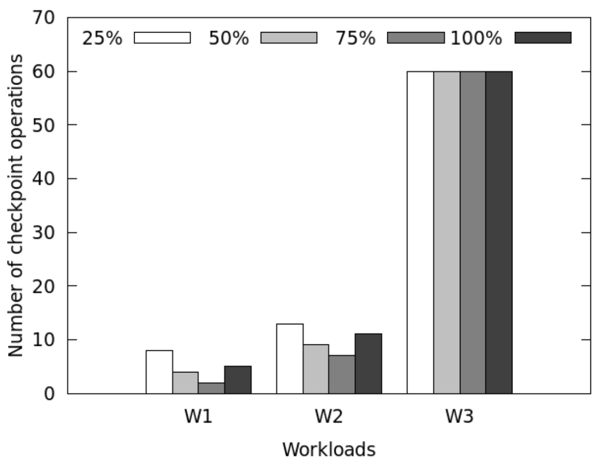

Fig. 8. The number of checkpoint operations of each workload under different checkpoint thresholds of NVRAM.

The number of journal writes in NAND flash storage was measured to further analyze the experimental results. Table I shows the number of journal writes in NAND flash storage under different checkpoint thresholds of NVRAM. In the cases of *W1* and *W2*, more journal writes were performed on the NAND flash storage when the checkpoint threshold of NVRAM was set to 100%. This is because, when a checkpoint operation is triggered while the NVRAM space is full, a commit transaction writes the updated blocks in the NAND flash storage regardless of the difference level. In the case of *W3*, the number of journal writes is much larger than that of *W1* and *W2* because the compression ratio of most updated blocks is expected to be low. Thus, the performance of *W3* is not affected by the checkpoint threshold of NVRAM. Due to these experimental results, the checkpoint threshold of DJ is set to 75%.

TABLE I
THE NUMBER OF JOURNAL WRITES IN NAND FLASH STORAGE UNDER DIFFERENT CHECKPOINT THRESHOLDS OF NVRAM

Workload	25%	50%	75%	100%
W1	1,205	1,198	1,204	24,681
W2	42,206	42,209	42,211	67,009
W3	263,434	263,427	263,414	263,408

C. *Experimental Results*

1) *Synthetic Workload*

Synthetic workloads were developed to evaluate the best and worst case performance of DJ. The synthetic workloads consist of 512 MB and 1,024 MB overwrites under different proportions of difference. As shown in Fig. 9, in the cases of the 10% and 90% differences, DJ outperforms ext4 by 16.8 and 2.3 times in the 512 MB and 1,024 MB overwrites, respectively. The performance improvement for the 512 MB overwrites is far higher because DJ does not trigger a checkpoint operation that would be caused by the lack of the journal area. This result indicates that DJ can delay a checkpoint operation by using the small-sized NVRAM efficiently. In the case of the 50% difference, which is the worst case scenario where there are no writes to the delta journal area, the performance of DJ is approximately the same as that of ext4. This result indicates that there is very little computing overhead to capture the differences.

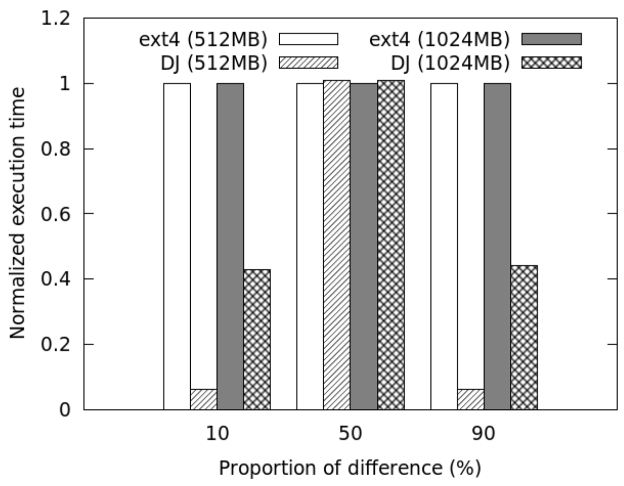

Fig. 9. Normalized execution time of synthetic workloads comparing ext4 and DJ. The execution time of DJ is normalized to that of ext4 according to the amount of overwrites.

2) *Micro Benchmark*

Postmark [26] is used as a micro benchmark. Postmark emulates an email server that concurrently performs read and write operations. The experiments were performed with 1,000 files, whose average size is 2 KB, by varying the number of transactions from 100 K to 500 K. Fig. 10 shows the normalized execution time of the Postmark benchmark, comparing ext4 and DJ. In the experimental results, DJ outperforms ext4 by up to 12.1% and 27.5% in the ordered and journal modes, respectively. This is because most small updates are stored as compressed deltas in the small-sized NVRAM instead of writing them to the NAND flash storage synchronously. Also, the performance of DJ in the journal mode is close to the performance of ext4 in the ordered mode. In general, the ordered mode is widely used in journaling file systems, even though the journal mode guarantees the highest data consistency. The experimental results show that the journal mode in DJ has adequate performance while providing the highest data consistency.

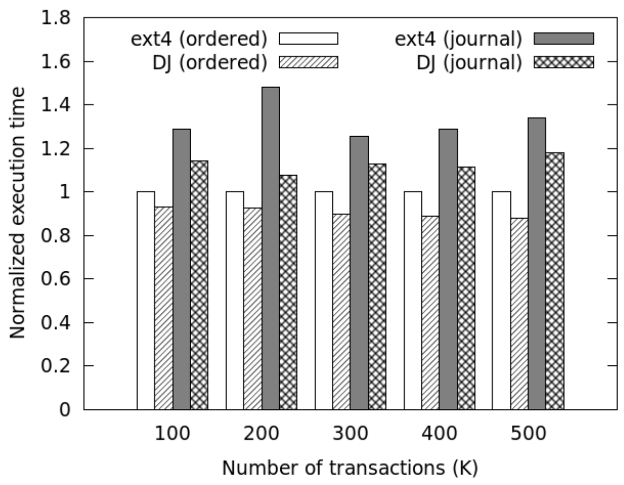

Fig. 10. Normalized execution time of Postmark comparing ext4 and DJ. The results are normalized to the ext4 ordered mode.

3) Real-World Workload

In the era of mobile computing, many popular applications manage their data using a lightweight database library [13]. Thus, the TPC-C benchmark with the lightweight database library is used for a more realistic workload on mobile devices. Fig. 11 shows the number of normalized transactions processed per minute (tpmC). The transaction throughput of DJ achieved improvements of 25.5% and 29.2% compared to that of ext4 in the ordered and journal modes, respectively. This is because most small updated blocks are stored as compressed deltas in the NVRAM, as shown in Fig. 12. In particular, no journal data is stored in the journal area of the NAND flash storage when DJ is mounted using the ordered mode. As a result, DJ can eliminate the I/O sync overhead and can thus improve transaction throughput. In the experimental results, the average compressed delta size per block is only 3.7% of the block size (i.e., about 150 bytes). This result indicates that DJ stores an updated block as a compressed delta in the NVRAM only when the compressed delta is small enough to efficiently use the small-sized NVRAM space.

In summary, DJ can improve the performance and the lifetime of NAND flash storage with only an additional 16 MB NVRAM by considering the characteristics of the update patterns in mobile devices. Furthermore, there is very little computing overhead (under 1%) when capturing the differences by using the host processor.

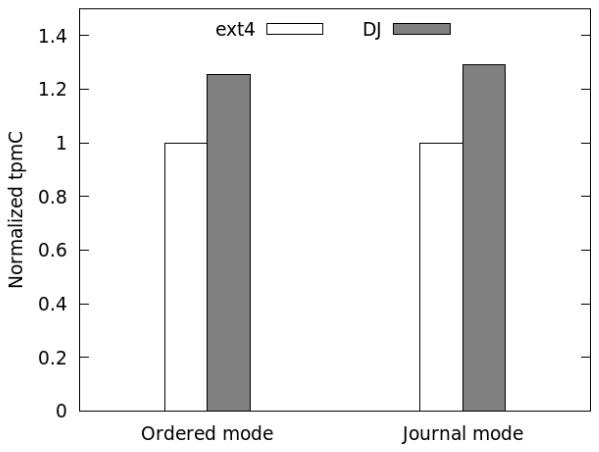

Fig. 11. Normalized transaction throughput of the TPC-C benchmark comparing ext4 and DJ. The transaction throughput of DJ is normalized to that of ext4 according to the journaling mode.

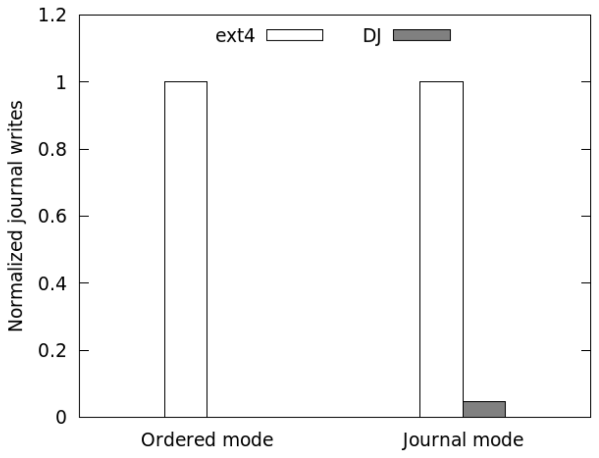

Fig. 12. Normalized journal writes of the TPC-C benchmark comparing ext4 and DJ. The number of journal writes of DJ is normalized to that of ext4 according to the journaling mode.

V. Conclusion

Journaling file systems are widely used in battery-powered mobile devices, such as smartphones and tablets, to guarantee data consistency. However, a traditional journaling technique degrades the performance and shortens the lifetime of NAND flash storage in mobile devices due to frequent storage accesses. Considering the increasing popularity of the journaling technique in mobile devices, excessive journaling overhead is a critical problem that needs to be addressed immediately. This paper proposed a novel journaling scheme that resolves this problem by using NVRAM efficiently. Although NVRAM, such as PCM or STT-MRAM, is emerging as a future storage device technology, its density and cost are not fully mature yet. Thus, the proposed scheme stores an updated block as a compressed delta in a small-sized

NVRAM only when the compressed delta is small enough. Experimental results showed that the proposed scheme can improve the performance and the lifetime of NAND flash storage with only an additional 16 MB NVRAM.

This work contributes to the analysis of the unique update pattern in mobile devices and presents a practical solution for improving I/O efficiency with only a slight increase in hardware costs. Also, the proposed solution can be easily applied to many different types of commercial mobile devices since it is designed to be compatible with traditional journaling file systems.

REFERENCES

[1] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems," *in Proc. USENIX Annual Technical Conference*, Anaheim, USA, pp. 105-120, Apr. 2005.

[2] V. Chidambaram, T. Sharma, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Consistency without ordering," *in Proc. USENIX Conference on File and Storage Technologies*, San Jose, USA, pp. 101-116, Feb. 2012.

[3] K. Lee and Y. Won, "Smart layers and dumb result: IO characterization of an Android-based smartphone," *in Proc. ACM International Conference on Embedded Software*, Tampere, Finland, pp. 23-32, Oct. 2012.

[4] S. Jeong, K. Lee, S. Lee, S. Son, and Y. Won, "I/O stack optimization for smartphones," *in Proc. USENIX Annual Technical Conference*, San Jose, USA, pp. 309-320, Jun. 2013.

[5] Y. Park and K. H. Park, "High-performance scalable flash file system using virtual metadata storage with phase-change RAM," *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 321-334, Mar. 2011.

[6] Y. Park and J.-S. Kim, "zFTL: Power-efficient data compression support for NAND flash-based consumer electronic devices," *IEEE Trans. Consumer Electron.*, vol. 57, no. 3, pp. 1148-1156, Aug. 2011.

[7] H. J. Choi, S.-H. Lim, and K. H. Park, "JFTL: A flash translation layer based on a journal remapping for flash memory," *ACM Trans. Storage*, vol. 4, no. 4, article 14, Jan. 2009.

[8] E. Lee, H. Bahn, and S. H. Noh, "Unioning of the buffer cache and journaling layers with non-volatile memory," *in Proc. USENIX Conference on File and Storage Technologies*, San Jose, USA, pp. 73-80, Feb. 2013.

[9] E. Lee, S. Yoo, J.-E. Jang, and H. Bahn, "Shortcut-JFS: A write efficient journaling file system for phase change memory," *in Proc. IEEE Symposium on Mass Storage Systems and Technologies*, San Diego, USA, pp. 1-6, Apr. 2012.

[10] D. Kim, E. Lee, S. Ahn, and H. Bahn, "Improving the storage performance of smartphones through journaling in non-volatile memory," *IEEE Trans. Consumer Electron.*, vol. 59, no. 3, pp. 556-561, Aug. 2013.

[11] 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 Trans. Consumer Electron.*, vol. 58, no. 3, pp. 825-833, Aug. 2012.

[12] G. Wu and X. He, "Delta-FTL: Improving SSD lifetime via exploiting content locality," *in Proc. ACM European Conference on Computer Systems*, Bern, Switzerland, pp. 253-265, Apr. 2012.

[13] W.-H. Kang, S.-W. Lee, and B. Moon, "X-FTL: Transactional FTL for SQLite databases," *in Proc. ACM International Conference on Management of Data*, New York, USA, pp. 97-108, Jun. 2013.

[14] H. Kim, N. Agrawal, and C. Ungureanu, "Revisiting storage for smartphones," *ACM Trans. Storage*, vol. 8, no. 4, article 14, Nov. 2012.

[15] G. Sun, Y. Joo, Y. Chen, D. Niu, Y. Xie, Y. Chen, and H. Li, "A hybrid solid-state storage architecture for the performance, energy consumption, and lifetime improvement," *in Proc. IEEE International Symposium on High Performance Computer Architecture*, Bangalore, India, pp. 1-12, Jan. 2010.

[16] J. K. Kim, H. G. Lee, S. Choi, and K. I. Bahng, "A PRAM and NAND flash hybrid architecture for high-performance embedded storage subsystems," *in Proc. ACM International Conference on Embedded Software*, Atlanta, USA, pp. 31-40, Oct. 2008.

[17] H. G. Lee, "High-performance NAND and PRAM hybrid storage design for consumer electronics," *IEEE Trans. Consumer Electron.*, vol. 56, no. 1, pp. 112-118, Feb. 2010.

[18] Y. Park, S.-H. Lim, C. Lee, and K. H. Park, "PFFS: A scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash," *in Proc. ACM Symposium on Applied Computing*, Fortaleza, Brazil, pp. 1498-1503, Mar. 2008.

[19] C. Lee and S.-H. Lim, "Efficient logging of metadata using NVRAM for NAND flash based file system," *IEEE Trans. Consumer Electron.*, vol. 58, no. 1, pp. 86-94, Feb. 2012.

[20] K. Kim, S.-W. Lee, B. Moon, C. Park, and J.-Y. Hwang, "IPL-P: In-page logging with PCRAM," *in Proc. Very Large Database Endowment*, vol. 4, no. 12, pp. 1363-1366, Aug. 2011.

[21] M. Rosenblum and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Trans. Computer Systems*, vol. 10, no. 1, pp. 26-52, Feb. 1992.

[22] C. Min, K. Kim, H. Cho, S.-W. Lee, and Y. I. Eom, "SFS: Random write considered harmful in solid state drives," *in Proc. USENIX Conference on File and Storage Technologies*, San Jose, USA, pp. 139-154, Feb. 2012.

[23] J. Kim, C. Min, and Y. I. Eom, "Reducing excessive journaling overhead in mobile devices with small-sized NVRAM," *in Proc. IEEE International Conference on Consumer Electronics*, Las Vegas, USA, pp. 19-20, Jan. 2014.

[24] H. Park, S. Yoo, and S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," *in Proc. Design Automation Conference*, San Diego, USA, pp. 59-64, Jun. 2011.

[25] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," *in Proc. Design Automation Conference*, San Francisco, USA, pp. 664-669, Jul. 2009.

[26] J. Katcher, "Postmark: A new filesystem benchmark," *Technical Report TR-3022, Network Appliance*, 1997.

BIOGRAPHIES

Junghoon Kim received his B.S. degree in Computer Engineering from Sungkyunkwan University, Korea in 2010 and M.S. degree in Mobile Systems Engineering from Sungkyunkwan University in 2012. He is currently a Ph.D. candidate in the Department of IT Convergence at Sungkyunkwan University. His research interests include storage systems, embedded systems, mobile platforms, and operating systems.

Changwoo Min received his B.S. and M.S. degrees in Computer Science from Soongsil University, Korea in 1996 and 1998, respectively, and his Ph.D. degree in Mobile Systems Engineering from Sungkyunkwan University, Korea in 2014. From 1998 to 2005, he was a research engineer in Ubiquitous Computing Laboratory (UCL) of IBM, Korea. Since 2005, he has been a research engineer at Samsung Electronics. His research interests include embedded systems, storage systems, and operating systems.

Young Ik Eom received his B.S., M.S., and Ph.D. degrees in Computer Science and Statistics from Seoul National University, Korea in 1983, 1985, and 1991, respectively. From 1986 to 1993, he was an associate professor at Dankook University in Korea. He was also a visiting scholar in the Department of Information and Computer Science at the University of California, Irvine, from Sep. 2000 to Aug. 2001. Since 1993, he has been a professor at Sungkyunkwan University in Korea. His research interests include virtualization, operating systems, cloud systems, and system securities.