Shortcut-JFS: A Write Efficient Journaling File System for Phase Change Memory

Eunji Lee, Seunghoon Yoo, Jee-Eun Jang

Dept. of Computer Engineering
Seoul National University
Seoul, Korea
{alicia0729, wing0cst, rofinsyer}@gmail.com

Hyokyung Bahn*

Dept. of Computer Engineering
Ewha University
Seoul, Korea
bahn@ewha.ac.kr

Abstract — **Journaling file systems are widely used in modern computer systems as it provides high reliability with reasonable performance. However, existing journaling file systems are not efficient for emerging PCM (Phase Change Memory) storage. Specifically, a large amount of write operations performed by journaling incur serious performance degradation of PCM storage as it has long write latency. In this paper, we present a new journaling file system for PCM, called Shortcut-JFS, that reduces write amount of journaling by more than a half exploiting the byte-accessibility of PCM. Specifically, Shortcut-JFS performs two novel schemes, 1) differential logging that performs journaling only for modified bytes and 2) in-place checkpointing that removes unnecessary block copy overhead. We implemented Shortcut-JFS on Linux 2.6, and measured the performance of Shortcut-JFS and legacy journaling schemes used in ext3. The results show that the performance improvement of Shortcut-JFS against ext3 is 40% on average.**

Keywords — ***Phase Change Memory; Journaling; File Systems***

I. INTRODUCTION

Reliability is one of the most important issues to be taken into account in the design of modern file systems. Specifically, as mobile devices such as smartphones and tablets are proliferating, sudden power failures occur more frequently and prompt recovery after a system crash is becoming increasingly important. To bring a file system to a consistent state, journaling schemes have been studied extensively in modern reliable file systems such as ext3 and ReiserFS [1]. This paper presents a new journaling file system efficiently designed for emerging PCM (phase change memory) devices.

PCM is a high-speed nonvolatile storage media that emerges recently, and there is a bright prospect that PCM will be used as secondary storage of computer systems like flash memory or hard disk [2, 27, 28]. This may be possible due to the rapid enhancement of micro-fabrication processes and multi-level cell (MLC) technologies [3, 4, 5]. It is expected that the cost of PCM will be no more than 3-5x of hard disk drive (HDD), and its power consumption will also be 10x lower than HDD.

However, there are several challenging issues in using current journaling file systems directly to PCM as physical characteristics of PCM are very different from those of hard disk or flash memory. To understand this, let us first see how journaling works. When a write request arrives, journaling file systems write data in the *journal area* first, which we call *write-ahead logging*, and then apply it to the original data location periodically, called *checkpointing*. This mechanism protects data from being corrupted in a sudden system crash because it always maintains consistent data either in the journal area or in the original location. Note that consistency may not be assured if we write data directly to the original location because the data may remain partially written when power failure occurs [7, 8].

Though journaling incurs more writes, it is efficient in hard disks as it reduces large seek overhead by writing sequentially to the journal area, changing write accesses to a sequential pattern. However, existing journaling designs may not perform well in PCM because PCM has no seek time but has relatively long write latency [6]. This implies that a large amount of writes incurred by journaling may result in serious performance degradations without achieving any advantages from sequential logging in case of PCM. Thus, journaling file systems need to provide an efficient handling of write operations rather than unnecessarily reduce seek overhead in PCM.

Motivated by this, we design a novel journaling file system called *Shortcut-JFS* that reduces write amounts by more than a half of existing journaling file systems considering the PCM characteristics, while providing high reliability.

To do this, we adopt two novel schemes in Shortcut-JFS. The first scheme is applied in the logging step. Shortcut-JFS writes only for changed bytes in the journal area, while legacy journaling schemes write an entire block even though only a few bytes are changed. This is possible because a write operation of PCM can be performed by the unit of a cell, which is different from HDD or flash memory that only allows writes for an entire block or page [9]. We call this scheme *differential logging*, which can reduce the write amount of logging significantly compared to existing journaling schemes.

Logged data in journal area are reflected to the original location of file systems by checkpointing where Shortcut-JFS uses another novel scheme to reduce writes. When applying the logged data to file systems, *Shortcut-JFS* does not copy the log to the original location. Instead, the log block itself

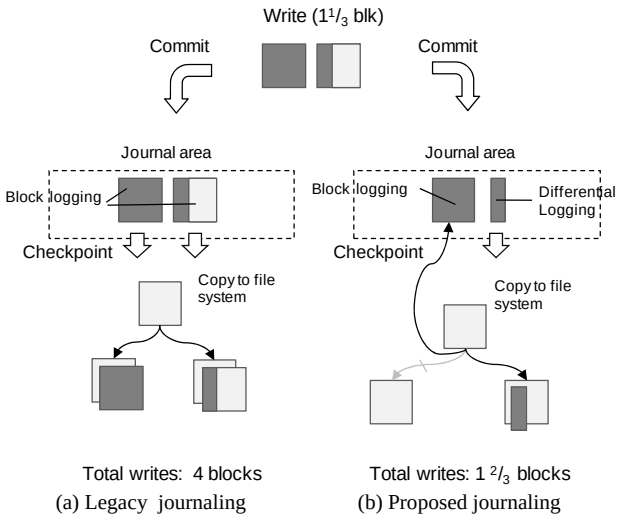


Figure 1. Comparison of journaling file systems

becomes a new location of the data block just by modifying a pointer to indicate it. Then, the physical location of journal area will be scattered as time goes on. This increases the seek time of storage seriously in case of HDD, but it does not influence the performance of PCM storage significantly. We call this scheme *in-place checkpointing*, which reduces the write amount of PCM significantly.

However, in-place checkpointing cannot be used when logged data is smaller than a block. The reason is that the logged data itself should be changed into the data block of file systems to perform in-place checkpointing. Thus, we use the differential logging scheme adaptively. To be specific, when an updated data size is larger than a half of a block, our scheme logs an entire block instead of using differential logging, which we call *block logging*. The reason we use block logging in this case is that *block logging* and *in-place checkpointing* totally needs one block write, while *differential logging* and *out-place checkpointing* requires more than one block writes.

Figure 1 compares the legacy journaling scheme and our Shortcut-JFS. When a write request of one and a third blocks arrives, the legacy journaling scheme writes totally four blocks, two blocks for logging and two blocks for checkpointing, while our Shortcut-JFS writes only one and two third blocks, one and a third blocks for logging and a third block for checkpointing, followed by a pointer update.

For performance evaluations, we implement Shortcut-JFS on Linux 2.6. Measurement results show that Shortcut-JFS improves the performance by 40% on average compared to ext3.

The remainder of this paper is organized as follows. Section II summarizes previous works on software techniques for PCM and Section III presents a new journaling file system for PCM storage, called Shortcut-JFS. Then, Section IV presents experimental results obtained from the implementation of Shortcut-JFS on Linux to assess the effectiveness of the proposed journaling file system. Finally, we conclude this paper in Section V.

II. RELATED WORKS

Considerable research has been performed on the efficient management of PCM when it is deployed in various storage hierarchy of computer systems.

Some research communities have studied on systems exploiting PCM as main memory [14-17, 24-26]. Mogul et al. suggested an efficient memory management policy for the hybrid memory system consisting of both DRAM and PCM. They proposed a page-attribute aware memory allocation policy that tries to place read-only pages like code segments in PCM, while load writable pages into DRAM, thereby reducing the amount of PCM writes [14]. In line with the research of Mogul et al., Querishi et al. proposed a memory architecture that uses a small amount of DRAM as a write buffer of PCM memory in order to prolong the lifetime of PCM and hide the long write latency of PCM [15]. Lee et al. also suggested a PCM memory architecture and attempted to improve the write performance between last level cache and PCM memory. They proposed two policies; buffer reorganization and partial writes, which track data modifications and write only modified cache lines or words to the PCM array [6, 16]. Lee et al. proposed the CLOCK-DWF algorithm for the hybrid memory architecture consisting of both DRAM and PCM [24]. They allocate read-intensive pages to PCM and write-intensive pages to DRAM, based on the characterization study of memory references. Zhou et al. suggested two wear-leveling techniques for PCM memory, that is row shifting and segment swapping, in order to prolong the lifetime of PCM [17]. Ipek et al. proposed *dynamically replicated memory* for PCM that maps two faulty physical pages into a single logical page, thereby reusing PCM pages that contain hard faults [8].

Another category of research focuses on the file system design for non-volatile RAMs including PCM. As the capacity of these non-volatile RAMs was small in early days, only a limited part of total file system image is located on the non-volatile RAM partitions. For example, Pramfs is designed to store frequently accessed or important data in non-volatile RAM, to support fast rebooting and resist crashes [18]. MRAMFS [19] and NEB file system [20] have been suggested to improve the space efficiency of non-volatile RAM based storage. MRAMFS saves space by compressing metadata, while NEB file system does this by extent-based file management.

As density of non-volatile RAM improves rapidly, recent studies focus on the design of a file system that keeps entire file system image on non-volatile RAMs. Baek et al. implemented a software layer to support both file objects and memory objects together in the unified memory system in which PCM serves as both main memory and storage [22]. Condit et al. redesigned the copy-on-write file system called BPFS for byte-addressable storage [7]. BPFS performs in-place write, when the updated data size is smaller than an atomic operation unit. This can reduce the outplace-update overhead of copy-on-write significantly. Wu et al. suggested a file system for storage class memory [23]. Assuming that storage class memory resides on the memory bus and can be

accessed directly from CPU, they proposed the file system that accesses files through the same address space of virtual memory systems.

Though extensive studies have been performed on PCM storage, we could not find researches on the journaling file system for PCM. To the best of our knowledge, our Shortcut-JFS is the first journaling file system that uses PCM as storage devices.

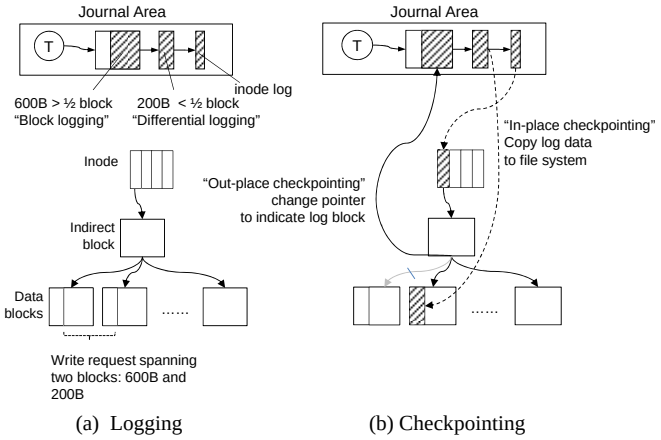
III. THE SHORTCUT-JOURNALING FILE SYSTEM

A. *Journaling Algorithm*

Figure 2 describes each step of write request handling processes in Shortcut-JFS with an example. For ease of explanation, we set the block size to 1KB and the inode size to 256B. In this example, a write request size is 800B spanning two different blocks; the fore 600B falls to the first block and the remaining 200B drops to the second block. This single write request is decomposed into a set of low-level writes that are two data block updates and one inode update for the metadata update such as last modified time. Shortcut-JFS handles a series of low-level write operations atomically and guarantees the file system consistency as follows.

First, it begins transactional writes for the given write request by creating a new transaction handler. To provide atomicity, we first set the transaction into the PENDING status. Then, we write the updated data into the journal area. At this point, we decide the journaling mode between differential logging and block logging according to the size to be updated. In this example, the first part is logged according to the block logging scheme because 600B is larger than a half of a block, while the second part is logged following the differential logging scheme as 200B is smaller than a half block. After logging data updates, inode updates need to be logged. In case of inode, Shortcut-JFS logs by the unit of an object for an easy management of uniform small objects.

As a result, three log chunks are logged in the journal area. Every chunk has its header and the headers are linked in the transaction list. Note that this example does not show details for brevity. After all updates for one transaction have been logged successfully, we change the transaction into the COMMIT state. Committed transactions are guaranteed to remain durable even after a system failure, and thus they can be reflected finally to the file system in any case. Figure 2(a) shows the layout of the journal area after committing the transaction.



(a) Logging (b) Checkpointing

Figure 2. Logging and checkpointing processes of Shortcut-JFS

The logged data in the journal area are finally reflected to the file system during checkpointing. Note that the checkpointing is performed periodically or activated when free space in the journal area is smaller than a certain threshold. Figure 2(b) shows the checkpointing process of Shortcut-JFS. We first scan all transactions committed after the last checkpoint, and flush their logged data to each location in the file system. In this process, Shortcut-JFS performs either in-place checkpointing or out-place checkpointing depending on the logged data size. As the first data chunk has the size of a block, Shortcut-JFS performs in-place checkpointing by modifying the pointer to indicate the log block. Then, the log block becomes a new location of the data block. In contrast, the second data chunk is smaller than a block, and thus Shortcut-JFS rewrites it to the original location in the file system. Finally, the inode object is written to the original location as it is also smaller than a block. After all logged chunks are checkpointed, the transaction status changes from COMMIT to CHECKPOINT. Journal area occupied by the checkpointed transactions are then reclaimed and become free.

Before concluding this section, let us now compare the total write amount of Shortcut-JFS with that of legacy journaling schemes used in ext3. Shortcut-JFS just writes 1.2KB (one block and 200B) while ext3 writes 4KB (two blocks for logging and two blocks for checkpointing).

B. *System Recovery*

Shortcut-JFS ensures the file system consistency and data safety against possible system failures. Here we discuss two cases of system failures to be considered primarily. First, a system crash can occur during logging. In this case, the file system remains as the last checkpoint state. In order to recover operations that have been committed in journal area after the last checkpoint, Shortcut-JFS performs *redo* operations by scanning the journal area from the last checkpoint position. In doing this recovery, we discard partially written transactions with PENDING status to prevent corrupted data from being

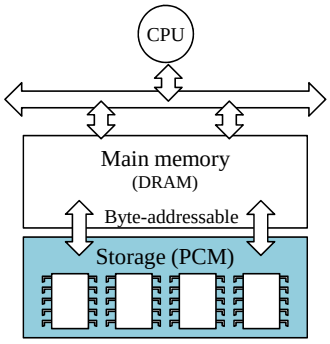


Figure 3. System architecture of PCM based storage

reflected to the file system. Second, it is possible that a system crash occurs during checkpointing. In this case, the file system itself can be corrupted as transactions that should be atomic might be partially reflected to the file system. However, as partially checkpointed transactions still remain in journal area, we can restore the file system into a consistent state. To do this, we scan the journal area from the last checkpointed position, and redo the checkpointing of committed transactions to the file system. In this way, Shortcut-JFS can recover corrupted file systems in a short time as it needs to scan only journal area, while file systems without journaling should scan all file system data blocks like fsck, incurring a long time of system recovery.

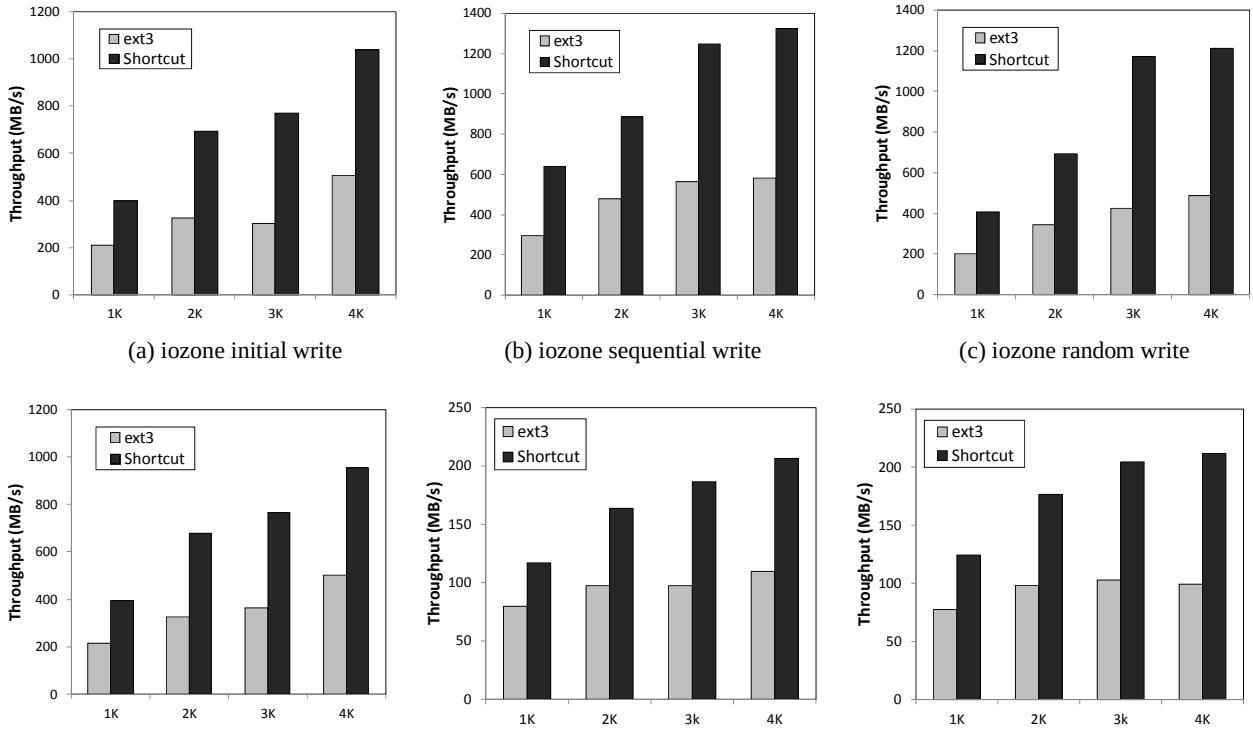
C. System Architecture

To reduce write amounts in PCM storage, we need an interface supporting byte accesses between main memory and storage devices. Though PCM is not commercialized as yet, most researches assume PCM to be placed in standard DIMM slots. This is a reasonable assumption as we need to utilize the byte-accessibility of PCM. However, this architecture has a weakness in that it makes the storage capacity limited by the number of DIMM slots. PCI express is another feasible option to place PCM to computer systems. A new standard interface that supports better access to PCM storage is also expected to appear soon [10]. We do not consider detailed architectures further as all interfaces aforementioned support byte-accessible property of PCM that is needed to implement Shortcut-JFS. Figure 3 shows a possible architecture of using PCM as storage medium in our study.

IV. PERFORMANGE EVALUATION

To assess the effectiveness of the proposed Shortcut-JFS, we have implemented Shortcut-JFS on Linux 2.6.32.24. To support byte-addressability of PCM, Shortcut-JFS is implemented by integrating journaling algorithms into ramfs [11], which is an in-memory file system using a part of in-memory buffer cache as storage. Since hardware platforms employing PCM storage do not exist yet, we alternatively use ram-disks backed by DRAM. To compare with our Shortcut-JFS, we also measure the performance of ext3 mounted on a ram-disk as ext3 is a representative journaling file system. In the experiments, we perform logging for data and metadata both. We use two well-known benchmarks, namely iozone [12] and postmark [13]. Iozone is a famous micro benchmark program to measure the file I/O performance, and postmark is one of the macro benchmarks that emulates an email server and web applications. We set the block size to 4KB, which is a common value in most operating systems and we commit updates to journal area in every 5 seconds according to the default configuration of ext3.

Figure 4 shows the throughput of ext3 and Shortcut-JFS for various configurations of iozone and postmark. For iozone, we perform experiments with four write scenarios that are initial write, random write, sequential write, and pwrite. Note that



(a) iozone initial write (b) iozone sequential write (c) iozone random write

(d) iozone pwrite (e) postmark fileset1 (f) postmark fileset2

Figure 4. Performance comparison of Shortcut-JFS and ext3

pwrite is a write scenario that supports pwrite system call. As postmark has its own write scenarios, we perform experiments for postmark by varying the fileset configurations. Fileset1 performs 1000 transactions on 1000 files of 1MB size and fileset2 does 10 transactions on 10 files of 50MB. Note that the total write amount of fileset1 and fileset2 reaches 500MB and 300MB, respectively.

To investigate the effectiveness of Shortcut-JFS under different write sizes, we measure the performance of file systems varying the write size from 1KB to 4KB. As shown in Figure 4, our Shortcut-JFS improves I/O performance by 2.1x and 1.8x on average for iozone and postmark, respectively. For all ranges of write sizes, Shortcut-JFS outperforms ext3. This result shows that Shortcut-JFS adaptively handles a various size of write operations efficiently. When a write size is less than a half of a block, Shortcut-JFS reduces write amounts by differential logging, while it enhances performance with the combination of block logging and in-place checkpointing when the write size is larger than a half block.

The performance improvement of Shortcut-JFS is larger in iozone than postmark. The reason is that postmark opens and closes a file for every single write operation, while iozone opens a file and makes a series of write operations in a batch way. Thus, the performance effect of reducing write amounts becomes larger in iozone than postmark.

To investigate the effectiveness of Shortcut-JFS precisely, we implemented four versions of journaling file systems based on Linux, which are block logging (BLK), differential logging (DIFF), block logging with in-place checkpointing (BLK-S), and adaptive logging with in-place checkpointing (ADP-S). The algorithms of these schemes are summarized in Table I. BLK uses the journaling scheme identical to that of ext3 and ReiserFS, and then our idea is incrementally added to DIFF, BLK-S, and ADP-S. Note that ADP-S represents our Shortcut-JFS. We compare the performance of these four configurations

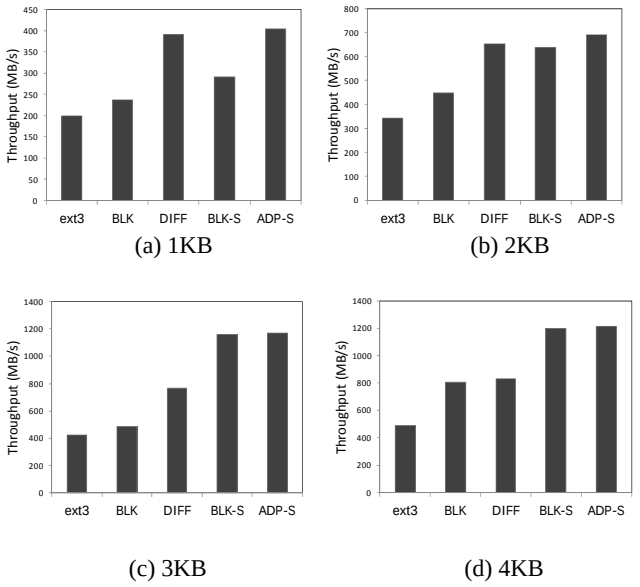


Figure 5. Throughput of different journaling file systems varying write size

TABLE I. SUMMARY OF JOURNALING ALGORITHMS

		BLK	DIFF	BLK-S	ADP-S
data	logging	blk	diff	blk	diff/blk
	checkpoint	rewrite	rewrite	switch	rewrite/ switch
metadata	logging	inode	inode	inode	inode
	checkpoint	rewrite	rewrite	rewrite	rewrite

with the original ext3. In the experiments, we measure the performance with different write sizes varying from 1KB to 4KB.

Figure 5 shows the throughput of each configuration with iozone benchmark. In the case of 1KB write size, DIFF and ADP-S performs well because they write differences only instead of an entire block. When the write size becomes a half block, that is 2KB, BLK-S and DIFF offers as good performance as ADP-S because these three schemes incur same amount of writes. Nevertheless, DIFF performs better than BLK-S. The reason we conjecture is that in-place checkpointing may incur scattering of data blocks, which generates more random writes. Though semiconductor memories are theoretically known to offer uniform performance, they are observed to have slightly slower random accesses than sequential accesses. When a write size is in the range of 3KB and 4KB, BLK-S and ADP-S show better performance than DIFF. That is because DIFF should perform write-twice for all updates while BLK-S and ADP-S writes at most one block. In summary, the performance of BLK, DIFF, and BLK-S varies depending on write sizes, but ADP-S provides consistently the best performance among the four versions of journaling regardless of write sizes.

One interesting result in the experiments is that, though current ext3 uses the same journaling scheme as BLK, BLK performs better than ext3 by a large margin. This performance gap comes from the software layers of ext3. Though both ext3 and BLK use ram-disk as a storage device, ext3 should pass through the block-device layer to access the file system in storage, while BLK maintains a persistent file system tree in the in-memory buffer cache layer. This result implies that software overhead also affects the I/O performance as much as the physical characteristics of storage devices, and our Shortcut-JFS achieves large performance enhancement with an efficient design of software techniques.

V. CONCLUSION

This paper presented a new journaling file system for PCM based storage systems, called Shortcut-JFS. Shortcut-JFS takes advantages of physical characteristics of PCM and reduces additional writes of legacy journaling significantly. Differential logging of Shortcut-JFS reduces write amounts significantly by logging only changed bytes instead of an entire block, and in-place checkpointing halves writes by reusing log blocks as file system data blocks. Shortcut-JFS is precisely designed considering overall system architectures and future byte-accessible interface of PCM. We implemented

Shortcut-JFS on Linux 2.6 and showed by measurement studies that it improves performance by 40% on average compared to ext3.

For future work, we plan to redesign our scheme considering wear-leveling and lifetime issues of PCM as PCM has limited write endurance.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.2011-0028825) (No.2011-0025633) and Ewha Global Top 5 Grant of Ewha Womans University. Hyokyung Bahn is the corresponding author of this paper.

REFERENCES

[1] V. Prabhakaran, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems," Proceedings of the USENIX Annual Technical Conference (ATC), 2005.

[2] R. F. Freitas and W. W. Wilcke, "Storage-class memory: the next storage system technology," IBM Journal of Research and Development, Vol. 52, No. 4, pp.439-447, 2008.

[3] C. D. Wright, M. M Aziz, M. Armand, S Senkader, and W Yu, "Can We Reach Tbit/sq.in. Storage Densities with Phase-Change Media?" Proceedings of the European Phase Change and Ovonics Symposium (EPCOS), 2006.

[4] F. Bedeschi et al, "A multi-level-cell bipolar-selected phase-change memory," Proceedings of the International Solid-State Circuits Conference, 2008.

[5] T. Nirschl et al. "Write strategies for 2 and 4-bit multi-level phase-change memory," Proceedings of the International Electron Devices Meeting, 2008.

[6] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," Proceedings of the 36th International Symposium Computer Architecture (ISCA), 2009.

[7] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger and D. Coetzee, "Better I/O through byte-addressable, persistent memory," Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP), 2009.

[8] E. Ipek, J. Condit, E. B. Nightingale, D. Burger, and T. Moscibroda, "Dynamically Replicated Memory: Building Reliable systems from Nanoscale Resistive Memories," Proceedings of the Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2010.

[9] B.-D. Yang et al. "A Low Power Phase-Change Random Access Memory using a Data-Comparison Write Scheme," Proceedings of the IEEE International Symposium Circuits and Systems (ISCAS), 2007.

[10] S. Venkataraman et al, "Consistent and Durable Data Structures for Non-Volatile Byte-Addressable Memory", Proceedings of the 9th USENIX Conference File and Storage Technologies (FAST), 2011.

[11] Ramfs, http://www.kernel.org

[12] W. Norcutt, the IOzone Filesystem Benchmark. http://www.iozone.org/.

[13] J. Katcher, "Postmark: a new file system benchmark," Technical report TR-3022, Network Appliances, 1997.

[14] J. C. Mogul, E. Argollo, M. Shah and P. Faraboschi, "Operating system support for NVM+DRAM hybrid main memory," Proceedings of the 12th workshop on Hot Topics in Operating Systems (HotOS), 2009.

[15] M. K. Qureshi, V. Srinivasan and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," Proceedings of the 36th International symposium on Computer Architecture (ISCA), 2009.

[16] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Phase Change Memory Architecture and the Quest for Scalability," Communications of the ACM, Vol. 53, No. 7, pp.99-106, 2010.

[17] P. Zhou, B. Zhao, J. Yang and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," Proceedings of the 36th International symposium on Computer Architecture (ISCA), 2009.

[18] PRAMFS: http://pramfs.sourceforge.net

[19] N. K. Edel, D. Tuteja, E. L. Miller, and S. A. Brandt, "MRAMFS: A Compressing File System for Non-Volatile RAM," Proceedings of the 12th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS), 2004.

[20] S. Baek, C. Hyun, J. Choi, D. Lee and S. H. Noh, "Design and Analysis of a Space Conscious Nonvolatile-RAM File System" Proceedings of IEEE Region 10 Conference (TENCON), 2006.

[21] T. Nirschl et al. "Write strategies for 2 and 4-bit multi-level phase-change memory," International Electron Devices Meeting, 2008.

[22] S. Baek, K. Sun, J. Choi, E. Kim, D. Lee and S. H. Noh, "Taking advantage of storage class memory technology through system software support," Proceedings of the workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA), 2009.

[23] X. Wu and A. L. N. Reddy. "SCMFS: A File System for Storage Class Memory," Proceedings of the International Conference on Supercomputing (SC), 2011.

[24] S. Lee, H. Bahn, and S. H. Noh, "Characterizing Memory Write References for Efficient Management of Hybrid PCM and DRAM Memory," Proceedings of the 19th IEEE/ACM International symposium on Modeling, Analysis, and Simulation of Computer and Telecomm. Systems (MASCOTS), Singapore, pp.168-175, 2011.

[25] N.H. Seong, D.H. Woo, and H.S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," Proceedings of the 37th International Symposium on Computer Architecture (ISCA10), pp.383-394, 2010.

[26] G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A hybrid PRAM and DRAM main memory system," Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC), pp.664-559, 2009.

[27] International Technology Roadmap for Semiconductors, Emerging Research Devices, 2007.

[28] Phase Change Memory: A new memory technology to enable new memory usage models, white paper, Micron Tech. Inc., http://www.numonyx.com/Documents/WhitePapers/Numonyx_PhaseChangeMemory_WhitePaper.pdf