MinL2R: Minimal Logging to Rollback-Recovery Scheme for Storage Class Memory

Hyun Ku Lee, Junghoon Kim, Dong Hyun Kang, and Young Ik Eom
College of Information and Communication Engineering, Sungkyunkwan University, Suwon, Republic of Korea
{hyungooo, myhuni20, kkangsu, yieom}@skku.edu

Abstract

Storage Class Memory (SCM) is being considered to be able to replace whole storage systems due to its own particular characteristics such as byte-addressability, high performance, and low power consumption. However, existing system may not utilize the advantages of SCM efficiently, because legacy file system is devised for relatively slow block-based storages such as hard disk drives. Especially, the consistency control mechanism for legacy file system incurs serious performance degradation of SCM storage. In this paper, we propose a novel byte-based consistency control scheme for SCM storage, called MinL2R. This scheme can reduce the amount of log data because it only records the original data to be modified, instead of all updates. Furthermore, when the overhead of data logging is higher than that of new block allocating, it only records block pointer to guarantee data consistency. Through our scheme, we can enhance the overall system performance without any loss of reliability. To evaluate the proposed scheme, we implement MinL2R on Linux 3.10.2 and measure the performance with I/O benchmarks. The experimental results show that, compared to EXT3 mounted on ram-disk, MinL2R enhances I/O throughput by 8 times on average.

1. Introduction

Storage Class Memory (SCM) such as PCM (phase-change memory), STT-MRAM (spin torque transfer magnetic RAM), and memristor is being considered as a candidate to replace whole storage systems. Although the large capacity of SCM is not produced due to its limited density and high cost, storage manufacturers and researchers regard SCM as a next-generation storage system because these problems may be solved in the near future. For example, Everspin announces a plan to increase the density up to a gigabit within the next few years [1]. Also, the cost of SCM is expected to be no more than 3-5 times to that of a hard disk [2]. Unlike block devices, SCM has particular character-istics such as byte-accessibility, zero-searching cost, and high performance. Considering the characteristics of the new storage devices, a new layout of storage system is required. First, we consider that SCM is connected to the memory bus directly. Existing storage systems access the storage device through generic block layer and emulate block I/O operations, that cause much overhead (e.g., I/O scheduling, emulation). Thus, SCM is connected directly to the memory bus in the proposed system, so we can access SCM with memory speed by eliminating unnecessary I/O stack. Another issue is the necessity of page cache (i.e., buffer cache). Page cache is effective to reduce the performance gap between memory and storage, even if the memory is just 8-10 times faster than SCM [3]. However, several developed features of SCM, especially STT-MRAM, have a bandwidth of up to 3.2 GB/s with nanosecond latency, which is in the same level as that of DRAM [4]. If memory and storage are under the same condition of access time, the maintenance of two copies makes unnecessary performance degradation. Thus, the proposed system eliminates page cache to reduce unnecessary copying overhead and enhance system performance.

Traditional block-based consistency control scheme causes significant write overhead in the SCM storage because this scheme writes in a block unit, though SCM device supports byte-access. Previous studies proposed various solutions to guarantee data consistency for SCM. However, they have several limitations. First, BPFS [5] proposed Short-Circuit Shadow Paging (SCSP), inspired by shadow paging, to reduce propagation of atomic updates, and new system layout without page cache. However, it requires additional hardware supports such as *epoch barrier*. Moreover, if the hardware supports of SCSP are

This work was supported by the IT R&D program of MKE/KEIT. [10041244, SmartTV 2.0 Software Platform]

implemented with software-based approach such as cache flush operation, BPFS makes excessive flush overhead. Second, Shortcut-JFS [6] adapts journaling scheme for PCM-based storage. However, since it is designed for page cache-based file system, it also makes excessive flush overhead for SCM storage as same as BPFS. Third, SCMFS [7] proposed write ordering process by using MFENCE and CLFLUSH instruction. However, it has no consistency control scheme for the data consistency.

In this paper, we propose a novel logging scheme, called MinL2R (Minimal Logging to Rollback-Recovery scheme), to reduce excessive consistency control overhead for the proposed system. MinL2R uses novel logging technique, called *minimal logging*, for system recovery, instead of journaling or shadow paging. This technique can reduce the amount of log data because it only records the proportion of original data, which will be modified with new data, instead of all updates. Furthermore, when the overhead of data logging is higher than that of new block allocating, it only records block pointer and writes new data to new block instead of original block for data consistency. Through this, MinL2R can greatly reduce the amount of data flushing and enhance the overall system performance without any loss of reliability.

The remainder of this paper is organized as follows. Section 2 presents the related work. In Section 3, we describe the detailed design of MinL2R. Section 4 describes the experimental results of MinL2R. In Section 5, we conclude this paper and suggest future directions.

2. Related Work

A number of file systems have been studied to support SCM. BPFS [5] suggested Short-Circuit Shadow Paging (SCSP) inspired by shadow paging to guarantee atomic update of metadata and data in program order. However, SCSP also has copying overhead to support atomic update of metadata and data. In addition, without hardware requirements (e.g., epoch barriers and 64bit atomic operation), the file system should implemented with software-based approach such as cache flush operation, thus it requires excessive flushing and caching overhead. In contrast, our scheme only flushes the proportion of original data to be modified for reducing those overheads. Shortcut-JFS [6] adapts journaling technique for PCM such as differential logging, which selects logging technique depending on the size of data. It is similar to our scheme but, since it is designed for page cache-based file system, it also makes excessive flush overhead for SCM storage. That is because it also should flush all of new data before updating the original data for consistency as same as BPFS. SCMFS [7] is a file system, which is built on virtual address to simplify the structure of file system. For consistency, the file system flushes metadata whenever it changed, but it flushes data to be modified periodically. Thus, it does not guarantee data consistency absolutely. In contrast, our scheme guarantees data and metadata consistency perfectly because we flush both data and metadata to be modified before updating for system recovery. On-demand Snapshot [8] suggests snapshot technique for versioning on BPFS. The snapshot technique reduces log size by using rollback-recovery and records block pointer for consistency. However, when a lot of small data writes are requested, this scheme incurs large performance degradation. That is because the scheme should allocate new blocks and copy all of the unmodified original data instead of writing new data to original data block directly. In contrast, our scheme records just original data to be modified when the log overhead is smaller than copy overhead. Thus, we can reduce copying overhead and memory usage when small size of data write is requested.

3. Design of MinL2R

3.1. Minimal Logging

MinL2R is devised for enhancing system reliability and performance based on SCM storage. Thus, our scheme uses novel logging technique optimized for SCM storage, called *minimal logging*. By using byte-addressable of SCM, *minimal logging* writes in a byte unit, instead of block unit, and consequently it reduces unnecessary bulk write. In addition, *minimal logging* uses rollback-recovery technique to minimize logging overhead. This technique records UNDO information instead of REDO information, in contrast with write-ahead logging and shadow paging. This technique reduces write overhead because the size of UNDO information is less than or equal to that of REDO information fundamentally. When we update original data with new data, write-ahead logging records all of new data for REDO, in case of (B) in Fig. 1. In contrast, rollback-recovery records overwritten area of original data for UNDO, in case of (A) in Fig. 1. As shown in Fig. 1, size of (A) is always less than that of (B) when we extent original data, because (B) is overlapped area between original data and (A). Furthermore, *minimal logging* chooses logging schemes depending on modified data ratio per block to minimize logging overhead. The ratio is represented as follows.

$$Modified\ data\ ratio = \frac{size\ of\ (B)\ in\ Fig.1}{size\ of\ original\ data} \quad (1)$$

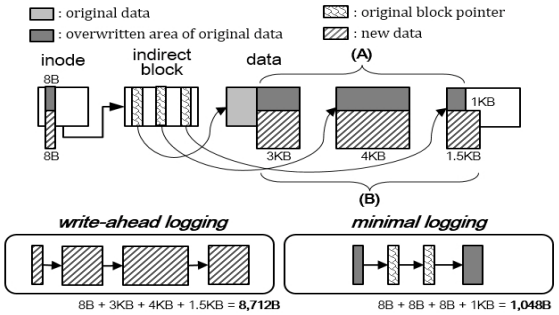

Figure 1. Comparison of consistency control schemes

Because the modified data ratio expresses the write overhead gap between copying of unmodified data and modified data, we set the threshold of the ratio as 50%. Thus, if the modified data ratio is smaller than 50%, *minimal logging* records original data, which is overlapped with new data, to logging area, we called this technique data logging. In this case, *minimal logging* writes new data to original block directly after logging process. If not, when the modified data ratio is larger than or equal to 50%, *minimal logging* executes pointer logging, which records only block pointer. In this case, *minimal logging* allocates new block and then fills the block with not overlapped with new data after logging process. As a result of Fig. 1, log data of MinL2R is smaller than that of write-ahead logging by 8 times, and the size gap is widened when the record size is increased.

3.2. Write Process.

To prepare write process, MinL2R creates logging header per I/O transaction, which includes inode number, status flag, and lists of log data. During the preparation of transaction, the status flag is set to *BEGIN*. After the preparation process, MinL2R executes *minimal logging* process, and sets up the status flag as *LOGGING*. During the process, *minimal logging* executes one of two logging techniques for consistency control: data logging or pointer logging. After *minimal logging* process, MinL2R executes *logged data flushing* process to finish the CPU cache sync. During the process, MinL2R uses cache flush operations such as *clflush_cache_range*, which update dirty data to memory and flush the cache line. After finishing all of logging process, *new data writing* process is executed. At the states, the status flag of transaction is changed to *WRITING*. During the process, MinL2R writes new data to SCM storage depending on the executed logging scheme per block. If the scheme is data logging, MinL2R writes new data to the original block directly. In sequence, If the scheme is pointer logging, MinL2R allocates new block and writes the new data to the new block. If there is no overlapped area in original data, MinL2R just allocates new block and writes new data to the block.

Meanwhile, if the logging area is filled with some of log data, and another writing operation is requested, we need more available space for log data. Thus, MinL2R finishes write process by force and then deletes the log data from logging area and old original data block. We call those procedures *checkout* process, which has *CHECKOUT* status. To reduce flushing overhead and guarantee that the new data is absolutely written to storage by file system, MinL2R performs *checkout* process when the available logging area is less than 75% and every 5 seconds as same as legacy consistency control technique.

3.3. Failure Recovery.

When the system failure occurs during write process, *failure recovery* process is executed depending on the status of the transaction. If write process has not begun, *failure recovery* process is not necessary because nothing have been changed in the original file. That is to say, the file system always guarantees the data consistency in spite of occurring the system failure, when the statuses of transaction are *BEGIN* and *COMMIT*. After *COMMIT* process, the original data is updated with the new data and the status of transaction is *WRITING* or *CHECKOUT*. If the system failure occurs during the states, *failure recovery* process is needed to restore the crushed file. During *failure recovery* process, MinL2R restores the crushed file depending on the type of log data. If the type of log data is *data rollback*, which recorded original data, MinL2R overwrites the crushed data with log data. If the type of log data is *pointer rollback*, which recorded block pointer, our scheme releases the new block and overwrites block pointer, which is in the indirect block, with the log data to restore the position of previous original data.

4. Evaluation

We implemented a prototype of MinL2R in kernel version 3.13.0. The experimental platform consisted of 3.4GHz Intel Core i7 CPU, 8GB of main memory. Since a large capacity SCM is not commercially produced yet, we used DRAM as a SCM storage in our experiment system. To measure the performance of MinL2R and legacy consistency control scheme, we used IOzone [9], the famous micro benchmark. For a baseline, we used ramfs, which uses DRAM as a storage without consistency control scheme. In order to compare our scheme with legacy block-based

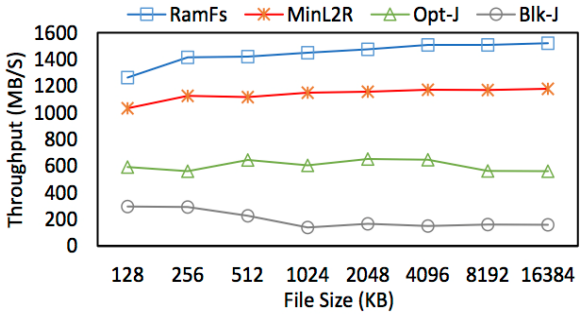

Figure 2. Write throughput of IOzone

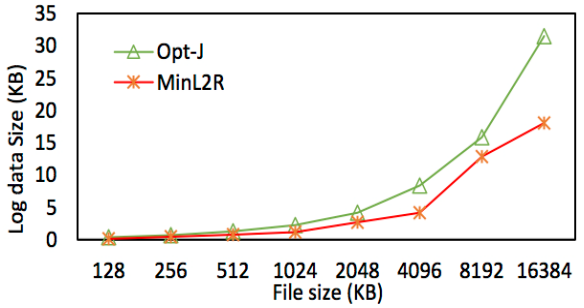

Figure 3. Log data size comparison

journaling on SCM storage, we mounted EXT3 file system on ram-disk with data journaling mode (Blk-J). Also, we implemented journaling scheme optimized for SCM with Shortcut-JFS consistency control scheme (Opt-J). Fig. 2 shows the write throughput comparison of MinL2R, Blk-J and Opt-J with IOzone. To investigate the effectiveness of MinL2R, we measured the performance of file systems varying the write size from 128B to 16KB with 1KB of record size. On average, MinL2R outperformed Opt-J from 165% to 210% under the different size of workloads. Moreover, MinL2R outperformed Blk-J from 2 times to 8.2 times. For all range, MinL2R outperformed other consistency control schemes with just 21% overhead than ramfs. Those results show that MinL2R handles various size of workloads efficiently.

To analyze the performance gap between MinL2R and other consistency control scheme, we measured the size of log data during the performance of benchmark. As shown in Fig. 3, log data size of MinL2R is lower than that of Opt-J. That is because MinL2R records just modified original data instead of all new data. Therefore, MinL2R reduces flushing overhead excessively and improves system performance.

5. Conclusion

In this paper, we proposed the system for SCM storage and novel consistency control scheme, called MinL2R. In the proposed system, we connect SCM to memory bus and access SCM storage with memory speed. Also, we eliminate page cache to reduce unnecessary copying overhead. Furthermore, MinL2R minimizes logging overhead by novel logging technique, called minimal logging. The technique chooses logging schemes depending on modified data ratio to minimize the size of log data. In addition, our technique reduces flushing and caching overhead by using UNDO information instead of REDO information. Experimental results showed that the proposed system greatly improve the overall I/O performance without any loss of reliability, by reflecting the byte-addressability and zero-searching cost of SCM. In future, we plan to explore efficient file management schemes for SCM storage and integrate with our proposed system.

6. References

[1] http://www.mram-info.com/everspins-ceo-explains-companys-technology- and- business.

[2] Freitas, R.F., Wilcke, and W.W., "Storage-class memory: The next storage system technology". IBM Journal of Research and Development 52(4.5), 2008, pp. 439-447.

[3] Lee, E., Jin, D., Koh, K., and Bahn, H. "Is buffer cache still effective for high speed pcm (phase change memory) storage?", Parallel and Distributed Systems (ICPADS), IEEE 2011, pp. 356-363.

[4] http://www.mram-info.com/everspin-officially-announces-worlds-first-st-mram-chip-will-be-available-2013.

[5] Condit, J., Nightingale, E.B., Frost, C., Ipek, E., Lee, B., Burger, D., and Coetzee, D.: "Better I/O through byte-addressable, persistent memory", ACM 2009, pp. 133–146.

[6] Lee, E., Yoo, S., Jang, J.E., and Bahn, H., "Shortcut-jfs: A write efficient journaling file system for phase change memory.", Mass Storage Systems and Technologies (MSST), IEEE 2012, pp. 1–6

[7] Wu, X., and Reddy, A "Scmfs: a file system for storage class memory", Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, ACM 2011, 39

[8] Lee, E., Jang, J., Kim, T., and Bahn, H. "On-demand snapshot: An efficient versioning file system for phase-change memory", Knowledge and Data Engineering, IEEE Transactions on, TKD 2013, pp. 2841–2853.

[9] Norcott., William D., Capps., and Don., "IOzone file system benchmark", http:// www.iozone.org