Implementation of Buffer Cache Simulator for Hybrid Main Memory and Flash Memory Storages*

Soohyun Yang and Yeonseung Ryu

Department of Computer Engineering, Myongji University
Yongin, Gyeonggi-do, Korea
sh871201@naver.com, ysryu@mju.ac.kr

Abstract. A buffer cache mechanism is usually employed in modern operating system to enhance the performance that is limited by slow secondary storage. In this paper, we present the implementation of a trace-driven simulator for buffer cache schemes that consider DRAM/PRAM hybrid main memory and flash memory based storages. The goal of simulator is to analyze the legacy buffer cache schemes by measuring the number of write operations on PRAM and the number of erase operations on flash memory.

Keywords: Simulator, Buffer Cache, Buffer Replacement, Non-volatile Memory, Flash Memory, DRAM/PRAM Hybrid Main Memory

1 Introduction

Most modern operating systems (OS) usually employ a buffer cache mechanism to enhance the I/O performance that is limited by slow secondary storage. When OS receives a read request from an application, file system in OS copies the data from storage to the buffer cache in the main memory and serves the next read operations from the faster main memory. Similarly, when OS services a write request, it stores data to the buffer cache and later flushes several data together to the storage. For the past decades, buffer cache schemes have been implemented for DRAM-based main memory and hard disk based secondary storage.

Recently, NAND flash memory is becoming important secondary storage for mobile computers because of its superiority in terms of fast access speed, low power consumption, shock resistance, high reliability, small size, and light weight. Further, recent studies have shown that DRAM-based main memory spends a significant portion of the total system power and the total system cost with the increasing size of the memory system [1]. Fortunately, various non-volatile memories such as PRAM (Phase change RAM), FRAM (Ferroelectric RAM) and MRAM (Magnetic RAM) have been developed as a next generation memory technologies. Among these non-volatile memories, PRAM is rapidly becoming promising candidates for large scale main memory because of their high density and low power consumption [2]. In order

* This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2010-0021897).

to tackle the energy dissipation in DRAM-based main memory, some recent studies introduced PRAM-based main memory organization [3] and DRAM/PRAM hybrid main memory organization [4, 5].

In this paper, we present an implementation study of buffer cache simulator that considers DRAM/PRAM hybrid main memory and flash memory storage devices. The goal of our simulator is to measure the number of write operations on PRAM and the number of erase operations on flash memory for several buffer cache schemes. Using our simulator, we can develop efficient buffer cache schemes to enhance the performance compared to existing schemes. The rest of this paper is organized as follows. Section 2 gives an overview of the non-volatile memory, flash translation layer and the previous buffer cache schemes. Section 3 explains the design of the simulator. Finally, Section 4 concludes the paper.

2 Background

2.1 Flash Memory and PRAM

A NAND flash memory is organized in terms of *blocks*, where each block is of a fixed number of *pages*. A block is the smallest unit of erase operation, while reads and writes are handled by pages [6]. Flash memory cannot be written over existing data unless erased in advance. The number of times an erasure unit can be erased is limited. The erase operation can only be performed on a full block and is slow that usually decreases system performance.

A PRAM cell uses a special material, called phase change material, to represent a bit. Table 1 shows the comparison of DRAM and PRAM. PRAM density is expected to be much greater than that of DRAM (about four times). Further, because the phase of the material does not change after power-off, PRAM has negligible leakage energy regardless of the size of the memory. Though PRAM has attractive features, the write access latency of PRAM is not comparable to that of DRAM. Also, PRAM has a worn-out problem caused by limited write endurance. Since the write operations on PRAM significantly affect the performance of system, it should be carefully handled.

Table 1. Comparison of DRAM and PRAM

Attributes	DRAM	PRAM
Non-volatility	No	Yes
Cost/TB	Highest (~4x PRAM)	Low
Read Latency	50 ns	50-100 ns
Write Latency	20-50 ns	~ 1us
Read Energy	~ 0.1 nJ/b	~ 0.1 nJ/b
Write Energy	~ 0.1 nJ/b	~ 0.5 nJ/b
Idle Power	~ 1.3 W/GB	~ 0.05 W
Endurance	∞	10^8 for write

2.2 Flash Translation Layer

If the flash memory is used as storage device, OS usually employs a software module called *flash translation layer* (FTL) between file system and flash memory device [7-11]. An FTL receives read and write requests from the file system and maps a logical address to a physical address in the flash memory. The address mapping schemes used in FTLs are classified into three groups depending on their granularity: *page-level*, *block-level*, and *hybrid-level*.

In the page-level scheme, a logical page number from the file system can be mapped to a physical page number in the flash memory. In the block–level scheme, the logical page address is divided into a logical block number and a page offset. The logical block number is used for finding a physical block that includes the requested page, and the page offset is used as an offset to locate the page in the corresponding block. Because of the disadvantages of the page-level and the block-level schemes, hybrid-level schemes have been widely used in the industry as a compromise between the page-level mapping and the block-level mapping. Most hybrid-level schemes use a log block mechanism for storing updates [8, 10, 11]. They divide the flash memory blocks into data blocks and log blocks. Data blocks represent the ordinary storage space, and log blocks are used for storing updates. The hybrid-level schemes maintain the block mapping table for the data blocks and the page mapping table for the log blocks. A major problem of the log block scheme is that it requires merge operations to reclaim the log blocks; this problem is explained further in the following sections.

A log block scheme, called block associative sector translation (BAST), was proposed in [8]. In the BAST scheme, flash memory blocks are divided into data blocks and log blocks. Data blocks represent the ordinary storage space and log blocks are used for storing updates. When an update request arrives, the FTL writes the new data temporarily in the log block, thereby invalidating the corresponding data in the data block. Whenever the log block becomes full or the free log blocks are exhausted, garbage collection is performed in order to reclaim the log block and the corresponding data block. During the garbage collection, the valid data from the log block and the corresponding data block should be copied into an empty data block. This is called a *merge operation*. After the merge operation, two erase operations need to be performed in order to empty the log block and the old data block. When the data block is updated sequentially starting from the first page to the last page, the FTL can apply a simple *switch merge*, which requires only one erase operation and no copy operations. That is, the FTL erases the data block filled with invalid pages and switches the log block into a data block.

2.3 Buffer Cache Schemes for Flash Memory Storages

The legacy flash aware buffer management schemes can be classified into two categories: page-level [12-15] and block-level schemes [16, 17].

In [12], a page-level scheme called CFLRU (Clean first LRU) was proposed. CFLRU maintains the page list by LRU order and divides the page list into two regions, namely the working region and clean-first region. In order to reduce the write cost, CFLRU first evicts clean pages in the clean-first region by the LRU order, and if there are no clean pages in the clean-first region, it evicts dirty pages by their LRU order. CFLRU can reduce the number of write and erase operations by delaying the flush of dirty pages in the buffer cache.

In [16, 17], block-level replacement schemes called FAB (Flash Aware Buffer management) and BPLRU (Block Padding LRU) were proposed, which consider the block merge cost in the log block FTL schemes. When a page in the buffer cache is referenced, all pages in the same block are moved to the MRU position. When buffer cache is full, FAB scheme searches a victim block from the LRU position which has the largest number of pages in the buffer cache. Then, all the pages of the selected block are passed to the FTL to flush into the flash memory. BPLRU scheme also evicts all the pages of a victim block like FAB, but it simply selects the victim block at the LRU position. In addition, it writes a whole block into a log block by the in-place scheme using the page padding technique. Therefore, all log blocks can be merged by the switch merge, which results in decreasing the number erase operations.

3 Buffer Cache Simulator

Fig. 1 illustrates the system configuration considered in our simulator in which main memory consists of DRAM and PRAM, and secondary storage is based on flash memory. We assume that the hybrid main memory consists of DRAM and PRAM, which are divided by a memory address. The memory which has the low memory address is DRAM and the high section is allocated to PRAM. We also assume that medium of storage device is flash memory. The flash memory model used in the simulation was the Samsung 16GB NAND flash memory [6]. The page size is 4 KB and the number of pages in a block is 64. We implement BAST scheme as an FTL scheme of flash memory because it is a representative and basic log block scheme. In BAST scheme, 100 log blocks were used.

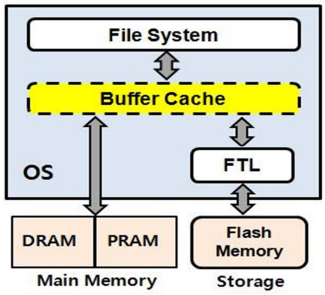

Fig. 1. System configuration for our simulator.

Our buffer cache simulator is a trace-driven simulator that use disk I/O traces as input. We extracted disk I/O traces from Microsoft Windows XP-based notebook PC, running several applications, such as document editors, web browsers, media player and games. The read/write ratio of workload is 67%/33%. We also made several synthesized disk I/O traces. Table 2 shows characteristics of synthesized traces examples used in our simulator. A read/write ratio "80%/20% in Table 2 means that the read and write operations in the traces are of 80 and 20 percentages, respectively. The locality in Table 2, e.g., 80%/20%, refers to that 80% of total references are performed in the 20% of storage device. We can also use I/O traces from an OLTP

application running at a financial institution [18] made available by the Storage Performance Council (SPC).

Table 2. Example of synthesized traces used in our simulator

Type	Total References	Read/Write Ratio	Locality
T8264	300,000	80%/20%	60%/40%
T8282	300,000	80%/20%	80%/20%

In order to simulate representative buffer cache schemes, we implemented some page-level buffer cache schemes like LRU and CFLRU, and some block-level schemes like BPLRU and FAB. When we allocate buffer to store data requested from file system, we allocate buffer from DRAM and PRAM alternately. Fig. 2 illustrates an example of block-based buffer cache structure that is implemented in the simulator.

Our simulator measures the hit ratio, the required number write operations on PRAM and the required number of erase operations on flash memory while varying the buffer cache size.

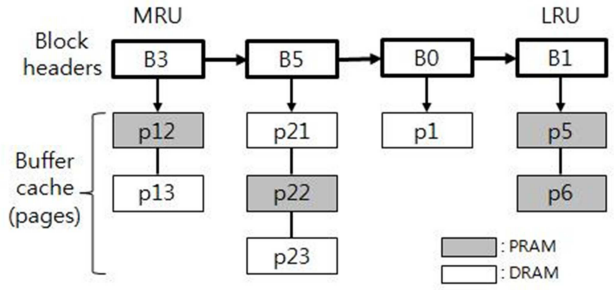

Fig. 2. Example of buffer cache structure in our simulator.

4 Conclusion

We have developed a simulator for several existing buffer cache schemes. In our buffer cache simulator, main memory consists of DRAM and PRAM, and secondary storage is flash memory. Our trace-driven simulator uses various kinds of disk I/O traces and measures performance metrics such as the hit ratio, the required number write operations on PRAM and the required number of erase operations on flash memory. By analyzing the behavior of legacy buffer schemes and their measured performance, we are going to design and implement novel buffer cache schemes to improve the performance.

References

1. L. A. Barroso and U. Holzle, “The Case for Energy-proportional Computing. Computer,” Vol. 40, No. 12, 2007.

2. X. Dong, N. Jouppi and Y. Xie, “PCRAMsim: System-level Performance, Energy, and Area Modeling for Phase-change RAM,” in Proc. of International Conference on Computer Aided Design, 2009.
3. M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in Proc. of International Symposium on Computer Architecture, 2009.
4. G. Dhiman, R. Ayoub, and T. Rosing, "PDRAM: A Hybrid PRAM and DRAM Main Memory System," in Proc. of Design Automation Conference, 2009.
5. H. Park, S. Yoo, and S. Lee, “Power Management of Hybrid DRAM/PRAM-based Main Memory,” in Proc. of Design Automation Conference, pp. 59-64, 2011.
6. Samsung Electronics, K9XXG08UXM.1G x 8 Bit/2G x 8 bit NAND Flash Memory
7. Gal and S. Toledo, “Algorithms and data structures for flash memories,” ACM Computing Surveys, vol. 37, no. 2, 2005.
8. J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho, “A space-efficient flash translation layer for compactflash systems,” IEEE Transactions on Consumer Electronics, vol. 48, no. 2, 2002.
9. Gupta, Y. Kim, and B. Urgaonkar, “DFTL: a flash translation layer employing demand-based selective caching of page-level address mapping,” in Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems, pp.229-240, 2009.
10. Y. Ryu, “SAT: switchable address translation for flash memory storages,” in Proc. of IEEE Computer Software and Applications Conference (COMPSAC), Jul. 2010.
11. Y. Ryu, “A Flash Translation Layer for NAND Flash-based Multimedia Storage Devices,” IEEE Transactions on Multimedia, 13, 3 2011.
12. S. Park, D. Jung, J. Kang, J. Kim, and J. Lee, “CFLRU: a replacement algorithm for flash memory”, in Proc. of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp. 234–241, 2006.
13. Y. Yoo, H. Lee, Y. Ryu, and H. Bahn, “Page replacement algorithms for NAND flash memory storages,” in Proc. of International Conference on Computational Science and its Applications, pp. 201-212, 2007.
14. Z. Li, P. Jin, X. Su, K. Cui, and L. Yue, “CCF-LRU: A new buffer replacement algorithm for flash memory,” IEEE Transactions on Consumer Electronics, Vol. 55, No. 3, pp. 1351-1359, August, 2009.
15. X. Tang and X. Meng, “ACR: An Adaptive Cost-aware Buffer Replacement Algorithm for Flash Storage Devices,” in Proc. of International Conference on Mobile Data Management, 2010.
16. H. Jo, J. Kang, S. Park, and J. Lee, “FAB: Flash aware buffer management policy for portable media players,” IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, pp. 485-493, 2006.
17. H. Kim and S. Ahn, “BPLRU: A buffer management Scheme for improving random writes in flash storage,” in Proc. of 6th USENIX Conference on File and Storage Technologies (FAST), pp. 239-252, 2008.
18. OLTP Trace from UMass Trace Repository. http://traces.cs.umass.edu/index.php/Storage/Storage.