Differentiated Space Allocation for Wear Leveling on Phase-Change Memory-based Storage Device

Soojun Im and Dongkun Shin, *Member*, IEEE

Abstract — *Phase-change memory (PCM) is the best candidate for the storage device of next-generation mobile consumer electronics. PCM has the potential to replace NAND flash memory, due to its non-volatility, in-place programmability, and low power consumption. Even though the lifetime of PCM is longer than that of flash memory, wear leveling is still required to cope with the non-uniformity of storage workload or malicious attack. In this paper, a novel wear-leveling algorithm for PCM storage is proposed, where more physical pages are allocated to frequently updated logical pages, to balance the wear counts of PCM cells. In comparison with the previous techniques, the proposed algorithm improved the lifetime of PCM by at maximum 14 times and on average 8 times*[1].

Index Terms — Phase Change Memory, Storage Class Memory, Wear-Leveling, Non-Volatile Memory.

I. Introduction

Mobile consumer devices generally use NAND flash memories to store persistent data. Recently, as a next-generation non-volatile memory, storage class memories (SCM), such as phase-change memory (PCM), spin-transfer-torque magnetic RAM (STT-MRAM), and resistive RAM (RRAM), are emerging to replace NAND flash memory [1]. SCM can preserve data without power, and consumes less power than DRAM does. In addition, SCM has shorter read and write latencies compared to NAND flash memory [2]. STT-MRAM is expected to even replace DRAM, due to its low read/write latencies, and high endurance. If main memory is non-volatile, there is no need to flush data in the main memory to another non-volatile storage before the device is turned off. However, PCM has longer read and write latencies than those of DRAM [1], [2]. Furthermore, PCM does not have enough endurance to be used as main memory. Therefore, PCM is expected to be employed as an alternative storage device for NAND flash memory, or as an auxiliary memory device in a hybrid memory architecture, where a small DRAM cache is backed by a larger capacity of PCM device [3], [4].

PCM has several advantages over flash memory. It supports byte-level (or word-level) read and write operations, while the flash memory requires page-level operations. In addition, PCM is over-writable, and thus it requires no garbage collection. Therefore, PCM is a proper storage device to store small-sized random data, such as file system metadata.

There were several studies on the exploitation of PCM as storage, where a PCM device was used as data storage [5], metadata storage [6], [7], [8], or as a write buffer for NAND flash memory storage [9], [10]. Although the endurance of a PCM cell is higher than that of a NAND flash memory cell, it is also necessary to make an effort to enhance the lifetime of PCM, since the storage workloads have non-uniform write patterns. Fig. 1 shows the cumulative portions of write amounts on all accessed sectors during the execution of two benchmark programs. For the storage access traces of smartphone storage and the metadata region of the FAT file system, a large amount of write requests are concentrated into only a small number of sectors. For example, more than 80% of write requests are concentrated on only 10% among all storage space accessed in the FAT metadata trace.

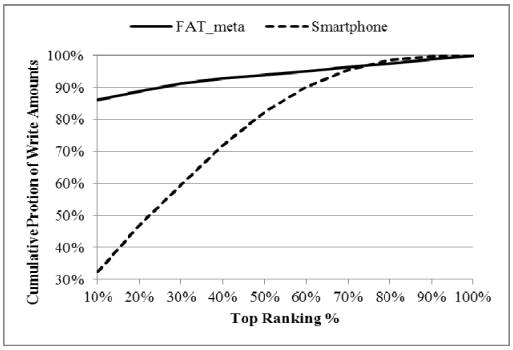


Fig. 1. Non-uniformity in write traffic

In particular, if PCM is used for a write buffer of flash memory storage, PCM requires much more endurance than flash memory. In addition, malicious attack can wear out PCM cells sooner than expected. Therefore, wear leveling is an indispensable function of PCM file systems, which distributes the write requests uniformly, by remapping heavily updated logical pages to less frequently updated physical pages.

The wear leveling techniques have been widely used for NAND flash memory devices, since they have a limited endurance. Generally, the flash translation layer (FTL) performs the wear leveling, through the logical-to-physical address remapping, which is required to handle the erase-before-write characteristic of flash memory [11], [12], [13]. The basic unit of wear leveling in FTL is an erase block. The

[1] This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (2010-0020724).

Soojun Im and Dongkun Shin are with the Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Korea (e-mail: lang33, dongkun@skku.edu).

Contributed Paper
Manuscript received 12/28/13
Current version published 03/20/14
Electronic version published 03/20/14.

wear-leveling technique maintains an erase count table in order to track the endurance of each flash memory block.

There are two types of wear leveling techniques. First, the allocation-based wear leveling tries to balance the wear outs of flash memory blocks, by allocating the least worn out block for write requests [14]. Second, when a less worn out physical block is not utilized by the allocation-based wear leveling, due to its cold data, the compulsory wear leveling moves the cold data to the most worn out block, and utilizes the less worn out block for write requests [15].

Generally, FTL can cause additional amount of writes over host requests, due to the garbage collection and compulsory wear leveling. That is called write amplification. The write amplification ratio (WAR) can be formulated as follows:

$$\text{WAR} = \frac{\text{data amount written by storage}}{\text{data amount written by host}} \quad (1)$$

Since the write amplification can degrade performance and overall endurance, it is important to balance the wear-outs of different blocks, without significant increase of the value of WAR.

In this paper, for PCM-based storage device, a novel wear leveling technique called differentiated space allocation (DSA) is proposed, which can balance the wear outs of PCM pages, while not invoking a large WAR. PCM requires no garbage collection, since PCM cells can be overwritten. Therefore, only the wear leveling technique affects the value of WAR. To balance the write counts of PCM pages, when a logical page is frequently updated by the host system, DSA allocates a greater number of physical pages in order to prevent further increment of the write counts of the allocated PCM pages. Therefore, all logical pages have a different number of allocated physical pages, depending on the update frequency. Experimental results showed that the proposed technique improved both the lifetime and write amplification ratio of PCM, compared with the previous PCM wear-leveling techniques.

The remainder of the paper is organized as follows. Section 2 introduces related works. Section 3 explains the proposed algorithm. Section 4 shows the experimental results. Finally, Section 5 concludes the paper.

II. Related Works

The technique for improving the lifetime of PCM device can be categorized into two groups: update minimizing and wear leveling. The update minimizing technique reduces the write counts on PCM cells. For example, the data comparison write (DCW) technique, proposed by Yang *et al.* [16], reads the old value before it is updated by a write request, and calculates the difference between the old data and new data. If the new and old values are the same at a bit location, DCW skips the programming for the corresponding bit cell. Cho *et al.* [17] proposed the Flip-N-Write technique, which also compares the new data and the old data, and checks whether the number of bit flips are more than half of the data size. If the condition is satisfied, Flip-N-Write inverts the input data in order to minimize the number of bit flips. Flip-N-Write guarantees the maximum number of bit flips never exceeds a half of the data size. Although DCW and Flip-N-Write can reduce the write count of each PCM cell, they cannot prevent wear out imbalance.

The PCM wear leveling technique uses a remapping mechanism in order to change the uneven host access pattern into a uniform access pattern on the physical PCM device. The remapping technique can be divided into table-based translation [18], and algebraic translation [19], [20]. The table-based translation algorithm maintains a logical-to-physical address mapping table, and write count table. Therefore, it has both memory and latency overheads by the mapping table.

The algebraic translation algorithm converts quickly between logical and physical addresses, by an algebraic mapping, without memory and latency overhead. Instead, it modifies the translation function periodically in order to balance the wear outs of physical pages. However, the WAR will be increased under the algebraic translation technique, because all data must migrate when the algebraic function is modified. In particular, even when most of the workloads are uniform except for small hot regions, the algebraic translation algorithm invokes unnecessarily data migrations, increasing WAR, because it has no information on the hot data regions. The table-based translation can minimize WAR by remapping only hot pages.

As an algebraic translation technique, Qureshi *et al.* proposed the randomized region-based start-gap (randomized RBSG) wear leveling technique [19], where the address space is partitioned into several segments, called RBSG regions. Each RBSG region has an extra storage line that allows the entire memory space within the region to be evenly worn out, by rotating each line one-by-one. Furthermore, each RBSG region has a Start pointer that points to a memory line with the lowest physical address in the region, and a Gap pointer that points to an empty line in the region. The mapping of lines from logical address to physical address is done by a simple arithmetic operation on Gap and Start pointers with the logical address. Start-Gap, however, must preserve an extra storage line to facilitate data movement, therefore space overhead is inevitable.

Seong *et al*. proposed another algebraic address translation algorithm, called security refresh [20], which swaps two randomly selected memory lines. Because the pair of swapping lines is periodically changed, security refresh is able to defend from malicious attacks.

Segment swapping [18] is a typical table-based translation algorithm, which periodically swaps segments with the maximum and minimum write accesses. Segment swapping needs a logical-to-physical address translation table, and a write count table.

Fig. 2 shows an example of segment swapping. There is a segment mapping table, which has the translation information between a logical segment number (LSN) and a physical segment number (PSN). The table also has the write count

(WC) of each physical segment. PSN 0 has the highest total write count, while PSN 2 has the lowest total write count. Therefore, it can be predicted that LSN 0 has hot data, and LSN 3 has cold data. To change the write count increasing rates of these two physical segments, the mapping can be changed. Then after the remapping, PSN 0 will be worn out slowly. However, since the swapping unit is a segment, the segment swapping technique cannot balance the write counts of different chunks within a segment. To mitigate the imbalance within a segment, the size of segment should be small. However, as the segment size becomes smaller, larger memory space overhead is required in order to manage the greater number of segments.

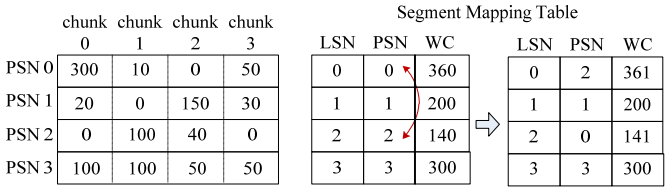


Fig. 2. Segment swapping

Our proposed DSA technique is also a table-based wear-leveling technique. Rather than swapping the physical spaces of hot and cold data, DSA allocates more physical space to hot data, to make even the write counts of physical PCM pages. Therefore, DSA performs the wear leveling more effectively with a smaller WAR.

III. Differentiated Space Allocation Wear Leveling

A. Overall Architecture

The main unit of the DSA wear-leveling technique is a segment. Therefore, an LSN-to-PSN mapping table is required. To reduce the required memory space for the mapping table, a large size of segment can be used. However, for hot segments, DSA manages the write count of each chunk within a segment, to mitigate the imbalance within the segment. DSA prevents increasing the write count of a chunk beyond the threshold value θ, by remapping the corresponding logical chunk to another physical chunk, if the original physical chunk is updated more than θ number of times.

Fig. 3 shows the overall architecture of DSA wear leveling. It is assumed that a segment consists of four chunks. Each chunk is represented by the pair of (segment number, chunk offset). Each LSN is mapped to a PSN. DSA manages the chunk-level write counts only for recently used segments. Therefore, the required memory space is small. In addition, DSA does not maintain the write counts of all segments, whereas the previous segment swapping technique manages the write counts of all segments. If the write count of a chunk in a recently used segment exceeds θ, a new physical chunk is allocated for the logical chunk from the *reserved segment pool* (RSP). The RSP is an overprovision area, the capacity of which is hidden to the host.

For example, in Fig. 3, LSN 1 is mapped to PSN 23, and the logical chunk (1, 2) is a hot chunk with the write count (N_{LW}) of 150. When the write count (N_{PW}) of the physical chunk (23, 2) reaches the threshold, 100, DSA allocates the physical chunk (100, 2) for the logical chunk (1, 2). The hot chunk mapping is written at the *hot chunk mapping table*. If N_{PW} of the additionally allocated chunk becomes θ, the chunk is changed into an expired chunk, and another chunk is allocated from the RSP. In Fig. 3, four physical chunks are exhausted by the logical chunk (0, 0), and the physical chunk (120, 0) finally has the up-to-date data. Therefore, each logical chunk consumes a different number of physical chunks, depending on the write frequency of the chunk. The physical chunks are prevented from being written over the value of θ, and thus the difference between the write counts of physical chunks is smaller than θ.

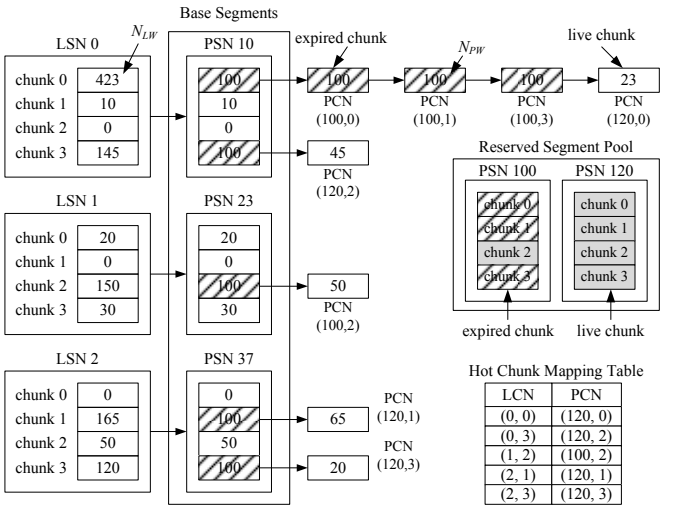


Fig. 3. Overall architecture of DSA wear leveling

B. Reserved Segment Replacement

When all segments in the RSP are exhausted, DSA randomly selects a base segment allocated for a logical segment, and replaces the oldest reserved segment in the RSP by the selected base segment. Since all the chunks in the oldest reserved segment are written frequently, it would be better to exchange the base segments written infrequently with the oldest reserved segment. However, DSA selects a victim base segment randomly, in order to remove the memory overhead required to track the least frequently written base segment. From the evaluation with real workloads, it is confirmed that the random-based victim selection policy does not show a significant degradation, compared to the optimal policy.

Before replacing the oldest reserved segment, all expired chunks should be de-allocated, and the data in live chunks must be moved to the base physical segment. The data of a live chunk in the oldest segment can be considered to be cold data, since the data is not updated frequently until the oldest segment is selected as a victim. For example, in Fig. 3, the physical segment with PSN 100 is a victim, and the chunk (100, 2) has a valid data. The data in chunk (100, 2) is moved

to chunk (23, 2), and the physical segment with PSN 100 is replaced by the physical segment with PSN 200, which was allocated for the LSN 150 as shown in Fig. 4.

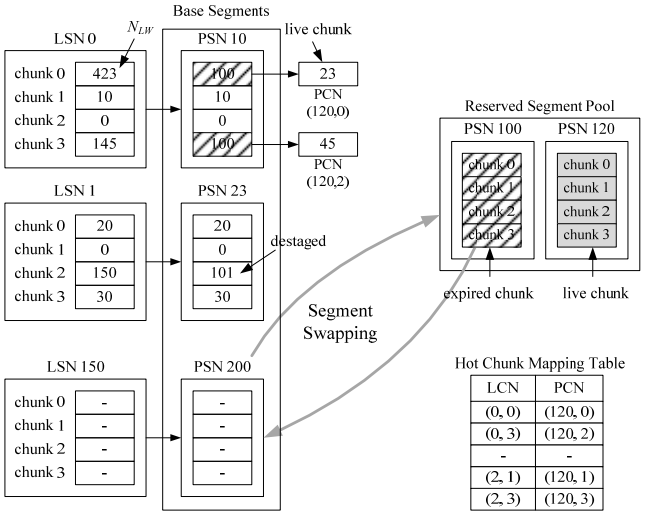


Fig. 4. Replacement of expired reserved segment

C. Hot Segment Detection

The DSA wear-leveling technique maintains the chunk-level write counts only for hot segments, and allocates reserved physical chunks when the number of write count of the hot logical chunk reaches the threshold value θ. To detect the hot segments, DSA manages the hot segment list, which has the chunk information of δ number of recently written segments, as shown in Fig. 5. When a write operation is requested by the host, DSA checks the hot list to find the corresponding segment. If the target segment is found in the hot segment list, the write count of the target chunk is increased. If the write count of the target chunk is the same as θ, DSA allocates a reserved chunk, and the hot segment list manages the write count of the newly allocated chunk. If the target segment is not found, the segment is inserted into the hot segment list. To insert the segment, the least recently written segment is replaced. The write count value of each chunk in the newly inserted segment is initialized to 0.

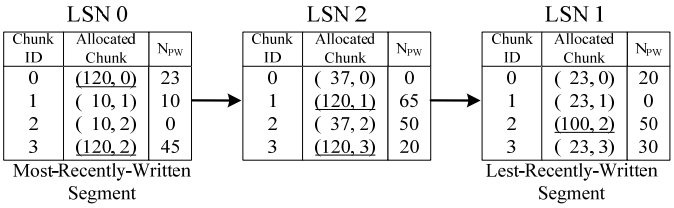


Fig. 5. Hot segment list

The hot chunk threshold θ, and the size of hot segment list δ, are important parameters to determine the performance of the DSA wear leveling algorithm. If θ is low and δ is high, the imbalance of write counts is mitigated, but the write amplification increases. In contrast, if θ is high and δ is low, there can be a large difference in write counts. Thus, it is necessary to decide appropriate θ and δ values, considering the trade-off between imbalance on wearing, and write amplification. If a target benchmark workload is given, it may be possible to determine proper θ and δ values.

D. Address Translation

To service host requests on a PCM storage device, DSA wear leveling must translate the logical chunk number into a physical chunk number. First, the logical segment number and the chunk offset are extracted from the logical chunk number. Second, the LSN and chunk offset are searched from the hot chunk mapping table, to check whether the target chunk is written at the reserved chunks. If the target chunk is not found, the base segment number is determined from the base segment mapping table. With the base segment number and chunk offset, the physical chunk can be accessed.

To minimize the searching overhead of the hot chunk mapping table, a hash list is used. The hash key is the chunk offset of the target chunk number. Fig. 6 shows the hash list. For each hash key, there is a linked list, which has the translation entries, with the hash key as chunk offset. For example, if the logical chunk number is 200, and a segment is composed of four chunks, the LSN is 50 (=200/4), and the chunk offset is 0 (=100 mod 4). Then, the translation entries of the hash key 0 are examined. In Fig. 6, there is the translation entry for the chunk (50, 0).

Under the hash list, the worst-case number of comparison is the number of reserved chunks. However, in most cases, the lists of different hash keys are balanced, and the target chunk is found among the recently written chunks. Experiments showed that the average number of comparisons is 1.1 times, when the device size, the segment size, the chunk size, and the reserved segment pool size are 128 MB, 512 KB, 8 KB, and 2 MB, respectively, for a benchmark trace.

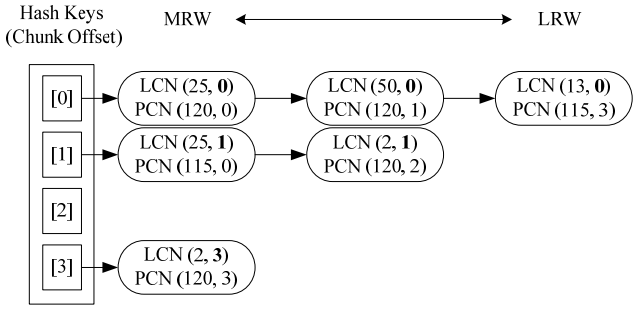


Fig. 6. Hot chunk hash list

E. Overhead

Compared with the segment swapping technique, the proposed DSA algorithm needs an additional reserved storage space, and a memory space for metadata. The required metadata are the logical-to-physical segment mapping table (L2PST), the hot segment list (HSL) shown in Fig. 5, and the hot chunk hash list (HCHL) shown in Fig. 6. The amounts of metadata are determined by several device parameters: device size (S_D), segment size (S_S), chunk size (S_C), number of reserved segments (N_R), and hot segment list size (δ).

Fig. 7 shows the change of metadata size, varying a device parameter with fixed remaining parameters. The fixed default values of S_D, S_S, S_C, N_R, and δ are 128 MB, 128 KB, 8 KB, 8, and 32, respectively. The metadata size is most sensitive to the segment size. As S_S is smaller, the size of L2PST is increased significantly. However, a large size of segment increases the size of HSL and HCHL.

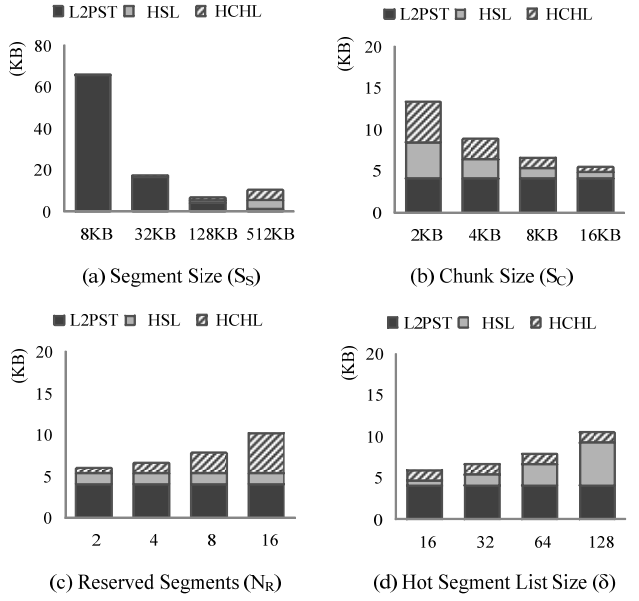


Fig. 7. Change of metadata size depending on the device parameters

IV. EXPERIMENTS

A. Experimental Environments

In order to estimate the performance of the DSA algorithm, and compare it with the segment swapping technique, a PCM simulator is implemented, which counts the number of update operations per sector unit (512 bytes). The size of the PCM storage device is assumed to 128 MB.

Four benchmark traces, Iozone, Iozone_Meta, Bonnie++, and Smartphone, are used. Iozone trace invokes many write operations on the metadata region of the file system. To know the performance of DSA when the PCM storage device is used for file system metadata storage, while a NAND flash memory is used for user data, only the write requests for the metadata are extracted from the Iozone trace, to make the Iozone_Meta trace. Smartphone trace was collected by executing several smartphone applications. Compared to other benchmark traces, Bonnie++ has a uniform write pattern, as shown in TABLE I. Bonnie++ trace is useful to check the amount of unnecessary write amplification by wear leveling.

The DSA algorithm is compared with the segment swapping technique, which is a typical table-based wear leveling algorithm. The maximum write count and the write amplification ratio of two wear leveling algorithms are compared. The maximum write count means the lifetime of the PCM device, and the write amplification ratio means the overhead of the wear leveling technique.

TABLE I
Benchmark summary

Benchmark	OS/ File system	Number of updates per sector		
		Max.	Std.Dev.	Avg.
Iozone	Linux/ext4	1,048,835	5,748	264
Bonnie++	Linux/ext4	20,642	126	476
Iozone_Meta	Linux/vfat	65,900	674	18
Smartphone	Linux/ext4	274,940	1,689	116

B. Performance Comparison

Fig. 8 and Fig. 9 show the write amplification ratios and the maximum write counts of the segment swapping technique, respectively, under various configurations. Four different segment sizes, 8 KB, 32 KB, 128 KB, and 512 KB, are used. The swap interval means the number of write operations between segment swapping operations, and was configured to one of 10, 100, 1000, and 10000. At each segment swapping, the data in the most worn segment and the least worn segment are swapped. To prevent the data moved by the previous swapping from being selected for the current swapping victim, the recently swapped segments are excluded from the swapping targets. For the segment swapping technique, the write amplification ratios are proportional to the segment size, and inversely proportional to the swap interval, in all benchmarks. For a 512 KB segment, the write amplification ratios are up to 66.2.

The maximum write counts of the segment swapping technique are significantly different, depending on the swap interval. For the Iozone benchmark, the maximum write count is high, when the swap interval is too short (10), or too long (10000). The short swap interval increases the overall write count, due to the high write amplification, and the long swap interval invokes the high maximum write count, due to the imbalance between segments. On the other hand, for the Bonnie++ benchmark, as the swap interval is long, the maximum write count decreases, since the trace has a uniform access pattern. The Iozone_Meta and Smartphone benchmarks show significant increases in the maximum write counts, as the swap interval increases. The I/O patterns of these benchmarks have many small random write requests. Therefore, frequent swapping is suitable.

Fig. 10 and Fig. 11 show the results of the proposed DSA algorithm under different θ and δ values. The values of S_S, S_C, and N_R are 128 KB, 8 KB, and 4, respectively. The DSA algorithm shows significantly low WARs, compared with the segment swapping technique. As θ is small and δ is large, many logical segments use the reserved segments, and thus the write amplification ratio increases. In particular, while Bonnie++ showed large WARs for the segment swapping technique, the values of WAR under the DSA technique are almost 1. This result means that the proposed DSA technique hardly increases the write amplification ratio, if the write pattern is uniform. In addition, Iozone and Bonnie++ benchmarks show similar WARs, for different θ and δ values.

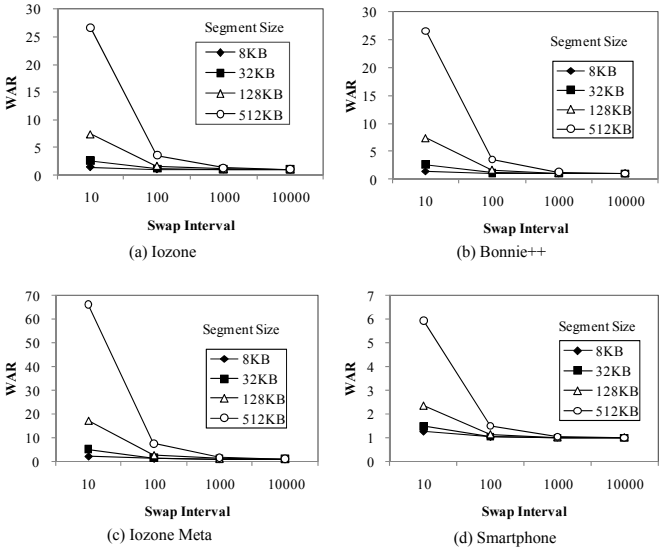


Fig. 8. Write amplification ratio under the segment swapping

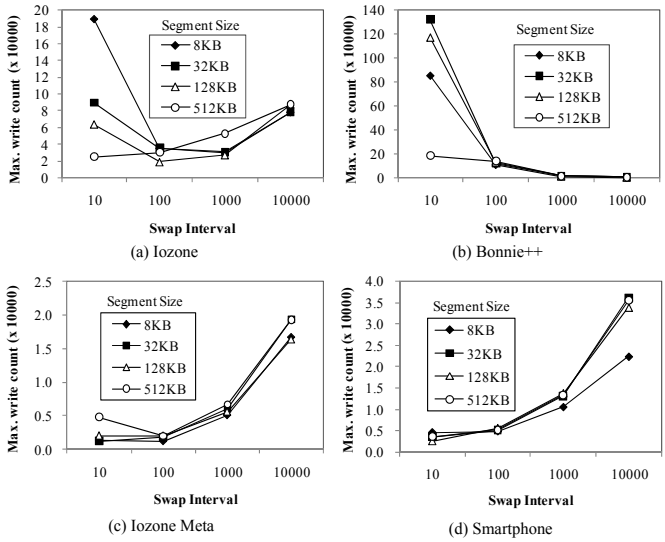


Fig. 9. Maximum write count under the segment swapping

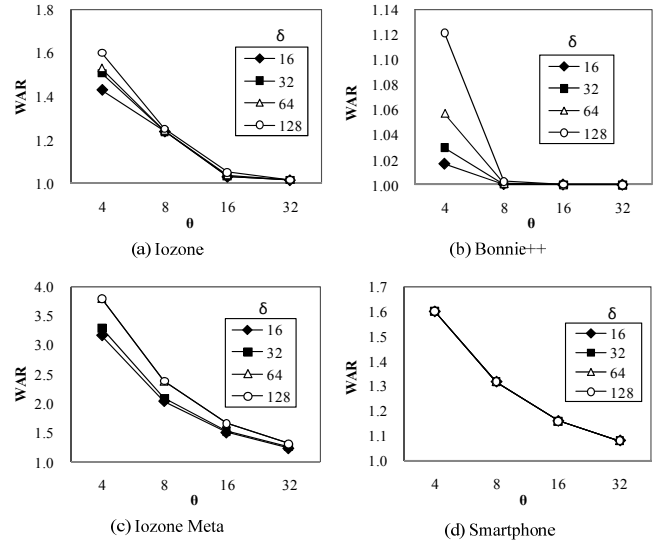


Fig. 10. Write amplification ratio under DSA

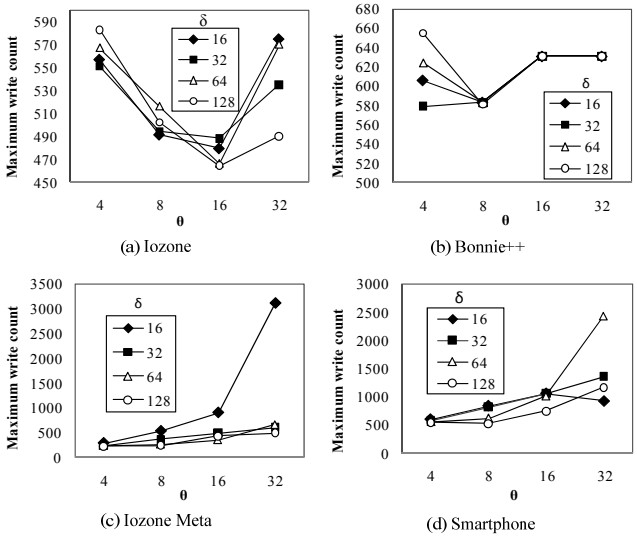
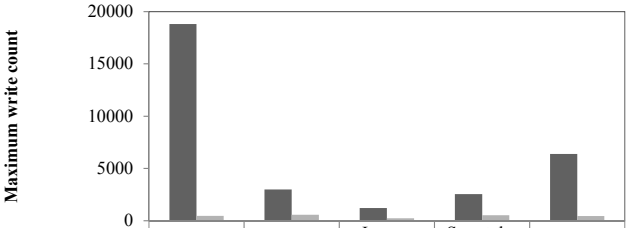


Fig. 11. Maximum write count under DSA

The maximum write counts by the DSA technique are under 0.3%, 0.05%, 15%, and 7% of those by the segment swapping technique for Iozone, Bonnie++, Iozone_Meta, and Smartphone, respectively. Therefore, it can be concluded that the DSA technique increases the lifespan of the PCM device, as well as minimizing the overhead of wear leveling.

Fig. 12 compares the maximum write counts of two wear-leveling techniques, under the best configurations of each technique. The DSA technique outperforms 14 times at maximum, and 8 times on average, compared with the segment swapping technique.



	Iozone	Bonnie++	Iozone Meta	Smartphone	average
segment swapping	18797	2992	1211	2553	6388.25
DSA	464	579	221	518	445.5

Fig. 12. Best results of segment swapping vs. DSA algorithm

V. CONCLUSION

In this paper, a novel wear-leveling algorithm for PCM-based storage device is proposed. The proposed DSA algorithm monitors the write frequency of recently written segments, and allocates more physical space to hot clusters for wear leveling. It can balance the wear outs of PCM cells with a small write amplification ratio. Moreover, in order to reduce the memory overhead of address translation, DSA uses dual-grained address mapping, which uses the chunk-level mapping only for the recently written segments. The simulation results showed that the proposed DSA algorithm increases the lifetime of PCM by a maximum of 14 times longer than Segment Swapping.

REFERENCES

[1] R. Freitas, and W. Wilcke, "Storage-class memory: The next storage system technology," *IBM Journal of Research and Development.*, vol. 52, issue 4.5, pp. 439-447, July, 2008.

[2] H. Volos, A. J. Tack, and M. Swift, "Mnemosyne: lightweight persistent memory," *in Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, California, USA, pp. 91-104, March, 2011.

[3] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable high performance main memory system using phase-change memory technology," *in Proc. ACM International Symposium on Computer Architecture*, Texas, USA, pp. 24-33, June, 2009.

[4] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mosse, "Increasing PCM main memory lifetime," *in Proc. Design, Automation and Test in Europe*, Dresden, Germany, pp. 914-919, March, 2010.

[5] J. Jung, Y. Won, and S. Kang, "Supporting block device abstraction on storage class memory," *in Proc. IEEE Workshop on Computing with Massive and Persistent Data*, Baltimore, MD, USA, pp. 1-4, Sep, 2008.

[6] C. Lee, and S. Lim, "Efficient logging of metadata using NVRAM for NAND flash based file system," *IEEE Transactions on Consumer Electronics.*, vol. 58, no. 1, pp. 86-94, 2012.

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

[8] 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*, Ceara, Brazil, pp. 321-334, 2008.

[9] 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.

[10] I. H. Doh, H. J. Lee, Y. J. Moon, E. Kim, J. Choi, D. Lee, and S. H. Noh, "Impact of NVRAM write cache for file system metadata on I/O performance in embedded systems," *in Proc. ACM Symposium on Applied Computing*, Hawaii, USA, pp. 1658-1663, March, 2009.

[11] J. Kim, J. M. Kim, S. H. Noh, S. Min, and Y. Cho, "A space efficient flash translation layer for compact flash systems," *IEEE Transactions on Consumer Electronics.*, vol. 48, no. 2, pp. 366–375, May 2002.

[12] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: locality-aware sector translation for NAND flash memory-based storage systems," *ACM SIGOPS Operating Systems Review.*, vol. 42, no. 6, pp. 36-42, Oct, 2008.

[13] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," *in Proc. ACM Architectural Support for Programming Languages and Operating Systems*, Washington, DC, USA, pp. 229-240, March, 2009.

[14] L. P. Chang, "On efficient wear-leveling for large-scale flash-memory storage systems," *in Proc. ACM Symposium on Applied Computing*, Seoul, Korea, pp. 1126-1130, March, 2007.

[15] Y. H. Chang, J. W. Hsieh, and T. W. Kuo, "Endurance enhancement of flash-memory storage, systems: an efficient static wear leveling design," *in Proc. ACM/IEEE Design Automation Conference*, California, USA, pp. 212-217, June, 2007.

[16] B.D. Yang, J. E. Lee, J. S. Kim, J. Cho, S. Y. Lee, and B. G. Yu, "A low power phase-change random access memory using a data-comparison write scheme," *in Proc. IEEE International Symposium on Circuits and Systems*, LA, USA, pp. 3014-3017, May, 2007.

[17] S. Cho, and H. Lee, "Flip-N-Write: a simple deterministic technique to improve PRAM write performance, energy and endurance," *in Proc. IEEE/ACM International Symposium on Microarchitecture*, New York, USA, pp. 347-357, Dec, 2009.

[18] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," *in Proc. ACM International Symposium on Computer Architecture*, Texas, USA, pp. 14-23, June, 2009.

[19] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling," *in Proc. ACM International Symposium on Microarchitecture*, New York, USA, pp. 14-23, Dec, 2009.

[20] N. H. Seong, D. H. Woo, and H. H. S. Lee, "Security refresh: prevent malicious wear-out and increase durability for phase-change memory with dynamically randomized address mapping," *in Proc. ACM International Symposium on Computer Architecture*, Saint-Malo, France, pp. 119-127, June, 2010.

BIOGRAPHIES

Soojun Im received the B.S. degree and M.S degree in computer engineering from Sungkyunkwan University, Korea in 2007 and 2010. He is currently a Ph.D. student in the School of Information and Communication Engineering, Sungkyunkwan University. His research interests include embedded software, general purpose graphic processing, memory management, file systems and flash memory.

Dongkun Shin (M'08) received the BS degree in computer science and statistics, the MS degree in computer science, and the PhD degree in computer science and engineering from Seoul National University, Korea, in 1994, 2000, and 2004, respectively. He is currently an associate professor in the Department of Computer Science and Engineering, Sungkyunkwan University (SKKU). Before joining SKKU in 2007, he was a senior engineer of Samsung Electronics Co., Korea. His research interests include embedded software, low-power systems, computer architecture, and real-time systems. He is a member of the IEEE.