Practical Issues in Designing of Garbage Collection for Solid States Drives

Taeho Nam and Taeseok Kim

Abstract—**Due to out-of-place update property of NAND flash memory, garbage collection is required for NAND flash memory based SSDs (Solid-State Drives). In this paper, we discuss some practical issues on developing the garbage collection scheme. First, both efficiency and longevity should be considered together for garbage collection design. To this end, we present simple and effective victim block selection, hot/cold page classification, and free block management schemes. Second, the spatial and temporal overhead of garbage collection should be small for being employed on real systems. Finally, we discuss how to make use of the internal architecture of SSDs for garbage collection design. We implemented new garbage collection scheme that considers these all on real SSD platform. Through extensive experiments, we show that the proposed garbage collection scheme provides a good efficiency as well as expands the longevity of SSDs.**

Index Terms—**NAND flash memory, garbage collection, solid state drives**

I. Introduction

Due to several desirable properties such as low data latency, low power consumption, and shock resistance, NAND flash memory based SSDs are widely used in various computing environments. SSDs use electronic interfaces compatible with traditional block I/O hard disk drives, but they employ an array of NAND flash memories instead of any moving mechanical components. Therefore, SSDs have a critical weakness of inefficient in-place update operations, which is inherited from characteristics of NAND flash memory.

To address this problem, flash memory based SSDs usually employ out-of-update operations [1]. The out-of-update is an operation that writes an updated data to a new free place, and leaves the place containing obsolete data as garbage. When there is not enough free space, the garbage space should be collected and then translated into free space. This procedure is called *garbage collection*.

In this paper, we discuss several practical issues on developing a garbage collection scheme for NAND flash memory based SSDs. First of all, garbage collection scheme should provide a good efficiency. Since garbage collection consists of many erase and write operations, it is important to reduce the number of erase and write operations as many as possible. Another important requirement for garbage collection design is to balance the erase count of all blocks. Since the erase count is limited to 100,000 for SLC (Single Level Cell) flash memory and 10,000 for MLC (Multi Level Cell) flash memory, respectively, the erase count should be carefully controlled.

In order to design a garbage collection scheme that satisfies two goals at the same time, we discuss the following four issues and present simple and effective policies, respectively.

1) How to select a victim block?
2) How to classify hot/cold pages?
3) How to manage the free blocks?
4) When or how often perform the garbage collection?

First, for victim block selection, we develop a simple model that evaluates each block by using the number of valid pages and erase count. Once a victim block is selected, all valid pages in the victim block should be copied into a free block. To improve the efficiency and longevity, we classify the valid pages into hot and cold by considering their update frequency and recency. Finally, hot valid pages are copied into old block and cold valid pages are copied into young block in order to wear out all blocks evenly.

Since other I/O commands are blocked during garbage collection, the spatial and temporal overhead of garbage collection should not be large. To this end, our victim block selection, hot/cold page classification, and free block management should not be complicated while improving both efficiency and longevity.

Finally, the internal architecture of SSDs can be used for garbage collection design. In SSDs with SATA interface, I/O commands from a host are queued in NCQ. Therefore, we can estimate the burstness of write workload by observing the status of NCQ. If NCQ is empty, we can perform garbage collection actively, if NCQ has many I/O commands, garbage collection is conservatively performed.

We implemented our garbage collection scheme on OpenSSD platform [2]. Through extensive experiments, we show that our scheme shows a good performance in terms of efficiency and longevity. In Section II, related works on garbage collection and the internal architecture of SSDs are presented. In Section III and IV, the proposed garbage collection scheme and experimental results are presented, respectively. Finally, we make a concluding remark in Section V.

II. Related Works and Background

A. The Existing Garbage Collection Schemes

When the free space is not enough in NAND flash memory,

Manuscript received January 30, 2013; revised March 15, 2013. 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 (2010-0010260).

The authors are with the department of computer engineering, Kwangwoon University, Seoul, 139-701, Republic of Korea (e-mail: thn7440@gmail.com, tskim@kw.ac.kr)

garbage collection is performed. The garbage collection first selects a victim block, copies valid pages in the block into a free block, and finally erases the victim block. In order to provide the efficiency, it is effective to select a block with many invalid pages as victim. That is because, as many invalid pages exist in victim block, more free space can be obtained by erasing the block. In addition, the operation time for garbage collection is reduced because a relatively small number of valid pages are just copied. Therefore, the number of invalid pages should be necessarily considered for high efficiency of garbage collection. On the other hand, the erase count of block should be also considered for victim selection. If a block with high erase count is selected as victim, it may affect the longevity of NAND flash memory. In conclusion, both longevity and efficiency should be appropriately considered for victim block selection.

Until now, many garbage collection schemes considering both longevity and efficiency have been proposed. Unlike the other schemes that will be introduced later, greedy algorithm only focuses on the efficiency of garbage collection [3]. Since it selects a block with the largest number of invalid pages as victim, it can be used as an indicator that provides an upper bound of efficiency. Cost Benefit (CB) algorithm is a first one that considers both efficiency and longevity [4]. It considers the longevity of NAND flash memory by using a value *a,* which means the elapsed time since the latest invalidation of a page in the block. Cost Age Time (CAT) is a similar approach with CB except that it uses the erase counts of blocks [5]. Hot/Cold Swapping (HC) technique that separates hot and cold data has been also proposed [6]. It periodically swaps the block with the largest erase count and the block with the smallest erase count for wear-leveling. It could increase the longevity of SSDs, but it has a significant swapping overhead. Turn-Based selection (TB) scheme has two phases; X and Y turn [7]. In X turn, TB selects a victim block using the greedy algorithm, while it selects a victim block according to some wear-leveling rules in Y turn. Static-Dynamic (SD) that employed in TrueFFS file system selects victims with a chain of physical flash blocks called Virtual Erase Unit (VEU) [8]. When garbage collection is needed, SD selects a VEU in a round-robin order. SD consists of dynamic and static phases for wear-leveling; in the dynamic phase, garbage collection is performed from the round-robin VEU queue, and in the static phase, hot-cold VEU swapping is performed. Dual-Pool (DP) performs data swapping between a young block and an old block similar to the Hot/Cold Swapping for wear-leveling [9]. Finally, Fast and endurant Garbage Collection (FeGC) has been proposed recently [1]. It considers the invalidated time of all invalid pages in a block for selecting a victim block and then redistributes valid pages by their update intervals. Moreover, FeGC manages free blocks efficiently and allocates them appropriately based on the characteristics of I/O workloads. By doing this, FeGC reduces garbage collection time and prolongs the lifetime of SSDs. They presented a significantly elaborate model for garbage collection, but the overhead is never low. It performs too many operations to select just one victim block and tries to classify all valid pages as hot or cold. For this reason, it may incur too large spatial and temporal overhead in large capacity of NAND flash memory based SSDs. As a matter of fact, our work has been started from this FeGC. We tried to minimize the temporal and spatial overheads of FeGC while providing a good efficiency and extending the longevity of SSDs.

B. SSD Architecture

In addition to the traditional considerations such as efficiency and longevity, we consider the architecture of SSDs for garbage collection design. To understand our approach that will be described later, it is necessary to carefully observe the internal architecture of SSDs. Fig. 1 shows an internal architecture of SSDs [2], [10].

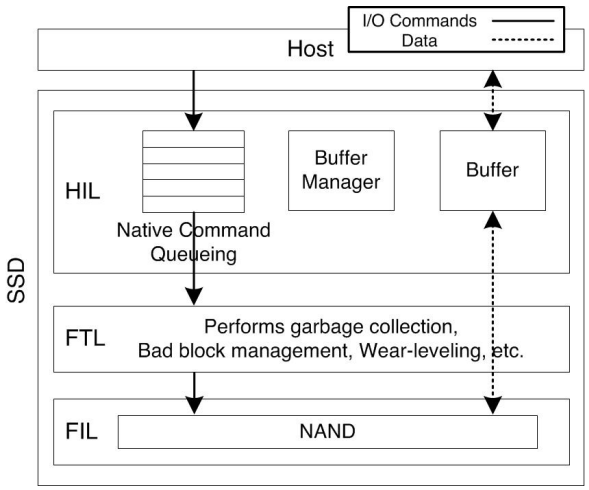


Fig. 1. An architecture of the SSDs.

The SSD can be largely divided into three layers. In the HIL (Host Interface Layer), I/O commands come from the host are queued in NCQ (Native Command Queuing). Once servicing a previous I/O command is completely finished, I/O command at the head of NCQ is dispatched. Finally, it is served from NAND flash memories via FTL (Flash Translation Layer). The main role of the FTL is to translate logical addresses to physical addresses. In addition, garbage collection, bad block management, and wear-leveling are performed in FTL. In this work, we focus on garbage collection in FTL, and make use of I/O commands information in NCQ for garbage collection design.

III. The Proposed Garbage Collection

In this section, we describe a new garbage collection scheme that considers the internal architecture of SSDs as well as longevity and efficiency. It is important to note that each procedure for garbage collection is designed with very low overhead, and thus it can be implemented on real SSD devices.

A. How to select a victim block?

As above mentioned, it is very important to select a victim block by considering both efficiency and longevity. To select the most suitable victim block, we calculate an evaluation value for each block.

$$\alpha \cdot V_i + (1-\alpha) \cdot E_i \quad (1)$$

Clock	1	2	3	4	5	6	7	8	9	HC
Page A				√		√		√		18
Page B					√		√			12
Page C	√	√	√							6
Page D									√	9

Fig. 2. The HC values of four pages.

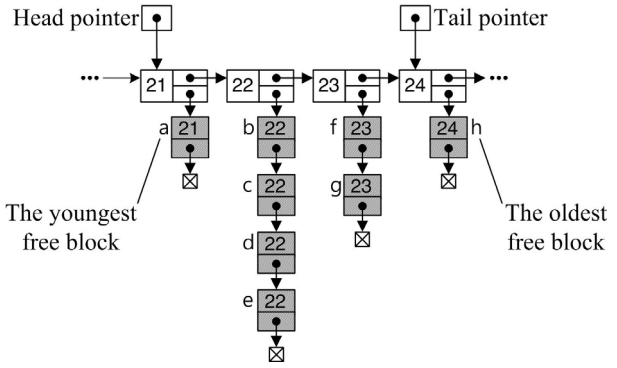


Fig. 3. The structure for free block management.

The value is calculated by (1), where V_i is the number of valid pages of ith block, and E_i is the erase count of the ith block. Then, a block with smallest evaluation value is selected as a victim. α is a constant between 0 and 1, which determines how much weight is put on efficiency or longevity. For example, when α is 0, only longevity is considered, and when α is 1, only efficiency is considered. Namely, by adjusting α, efficiency and longevity can be considered with an appropriate portion.

In order to implement this victim block selection policy, it is sufficient to maintain only the number of valid pages and the erase count for each block. Moreover, selecting a victim can be performed in a constant time if all blocks are mainlined in the evaluation value order.

B. How to classify Hot/Cold Pages?

To increase efficiency and longevity of garbage collection, we classify all valid pages into hot and cold. After the classification of hot/cold pages, hot pages are copied into young block that has low erase count, and cold pages are copied into old block that has high erase count. The hot/cold page classification can be performed based on their update frequency and recency. In order to consider both frequency and recency of page update, we develop a simple and effective technique as follows.

We associate each page with a HC (Hot/Cold) value that indicates its hot degree by analyzing update frequency and recency. We also maintain a clock, which is incremented for every page update. Whenever an update to a page is made, the value of clock is added to the HC value of the page. Fig. 2 illustrates an example that calculates the HC values of four pages. In this example, the HC values of page A, B, C, and D are 18, 12, 6, and 9, respectively.

Now, it is trivial to determine if a specific page is hot or cold. If the HC value of a page is higher than average of all valid pages, the page can be classified as hot, otherwise, can be classified as cold. In Fig. 2, since average of all HC values is 11.25, page A and B are hot pages, and page C and D are cold pages, respectively.

In order to implement this hot/cold classification policy, the HC values of all valid pages and only one average value are maintained. Since these values are calculated only when page is updated, the computation cost is also very low. During garbage collection, it is sufficient to just compare the HC value of each valid page and the average value.

C. How to Manage the Free Blocks?

As mentioned in Section I, since the erase count is limited in SSDs, all blocks should be controlled to evenly worn out. To this end, we copies hot pages into young block and cold pages into old block like FeGC [1]. Fig. 3 illustrates data structure for free block management. There are several shaded node lists that contain a group of free blocks with the same erase count. The shaded node lists are also maintained in a linked list in ascending order of erase count. During garbage collection, if hot valid pages should be copied back, the oldest block is allocated for them. On the contrary, if cold valid pages should be copied back, the youngest block is allocated.

The spatial overhead in the free block management is not large. In the worst case, namely, all free blocks have their own different erase counts, this scheme needs (2 * the number of free blocks) nodes. However, the situation does rarely happen because the wear-leveling is considered in both victim block selection and hot/cold page classification. Since the youngest block and oldest block can be retrieved from head and tail pointers, respectively, the temporal overhead during garbage collection is also very low.

D. When or How Often Perform the Garbage Collection?

In SSDs with SATA interface, I/O commands from a host are queued in NCQ, and an I/O command at the head of NCQ is handled first. If this simple observation is used for garbage collection design, we can solve the problem of when and how often garbage collection should be performed. If NCQ is empty, the garbage collection may be continuously performed until enough free space is obtained. When the free space is enough, if many write commands unexpectedly rush into SSDs from host, lots of write operations can be handled without performing the garbage collection for some time. For all that, garbage collection should not be performed needlessly. If a victim block has few invalid pages, the gain obtained by performing the garbage collection would be small.

On the other hand, if there are many I/O commands in NCQ, it is good to delay the execution of garbage collection as late as possible in order to provide a fast response time for each I/O command. Of course, the garbage collection should be performed when there is no free space, even if NCQ is not empty.

IV. Evaluation Results

To show the effectiveness of the proposed garbage collection scheme, we implemented it on OpenSSD platform [2]. OpenSSD platform is based on the commercially successful controller, and it is used to develop the SSD firmware. We compared our garbage collection with greedy algorithm that selects a block with the largest invalid pages as

victim. In our victim block selection policy, since α determines how much weight is put on efficiency or longevity, we experimented by varying the value of α.

To generate I/O workloads, we used the IOmeter benchmark tool that is an I/O subsystem measurement and characterization tool [11]. In our experiments, the SSD capacity is 6GB and 26GB of data are written to the SSD with a random pattern.

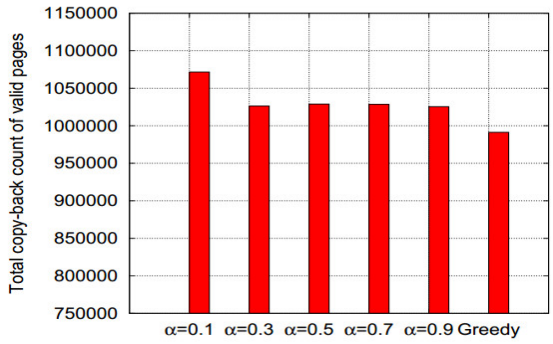


Fig. 4. Total copy-back count of valid pages.

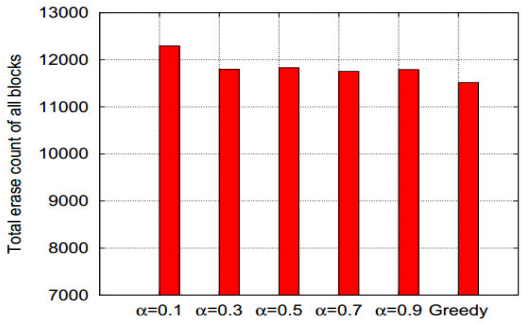


Fig. 5. Total erase count.

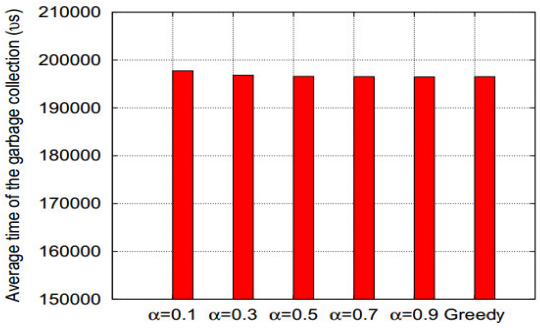
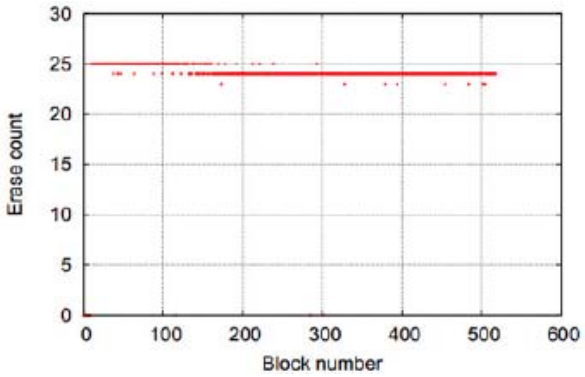


Fig. 6. Average time of the garbage collection

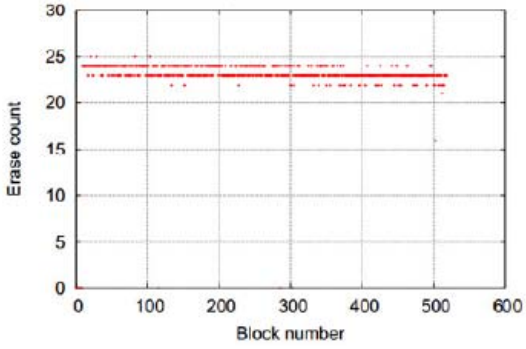
TABLE I: SSDs BENCHMARK RESULTS.

Scheme	Sequential read	Sequential write	4K Random read	4K Random write	Score
α=0.1	64.57 MB/s	17.93 MB/s	10.01 MB/s	1.74 MB/s	47
α=0.3	64.56 MB/s	18.80 MB/s	10.01 MB/s	1.77 MB/s	47
α=0.5	64.57 MB/s	18.47 MB/s	10.02 MB/s	1.78 MB/s	47
α=0.7	64.54 MB/s	18.79 MB/s	10.01 MB/s	1.78 MB/s	47
α=0.9	64.53 MB/s	18.88 MB/s	10.02 MB/s	1.78 MB/s	47
Greedy	64.54 MB/s	17.61 MB/s	10.02 MB/s	1.79 MB/s	47

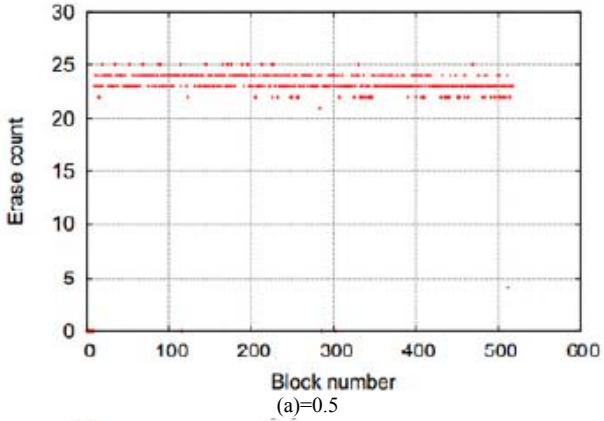
Fig. 4 and 5 shows the total copy-back count of valid pages and the total erase count that occurred during garbage collection, respectively. Since these two metrics shows the number of operations required during garbage collection, we can say that they indicate the efficiency of garbage collection. As can be seen, two results exhibit similar behavior. In both results, the greedy algorithm has the smallest counts, and our scheme shows a little larger counts. When α is close to 0, longevity are much considered than efficiency, and thus both copy-back count and erase count increase as the value of α decreases. However, even when α is 0.1, the proposed scheme shows just about 6% larger count than greedy algorithm in both results. Note that the greedy algorithm considers only efficiency. Therefore, we think that the 6% gap is never large.



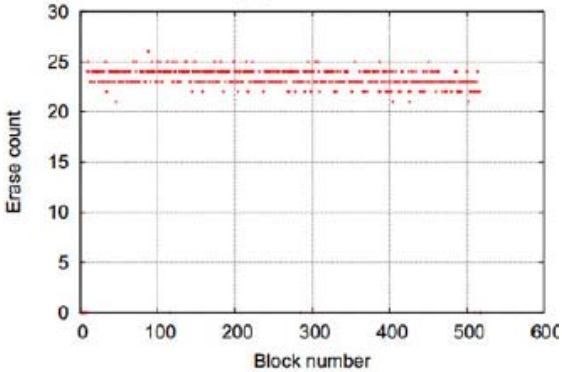
(a)=0.1



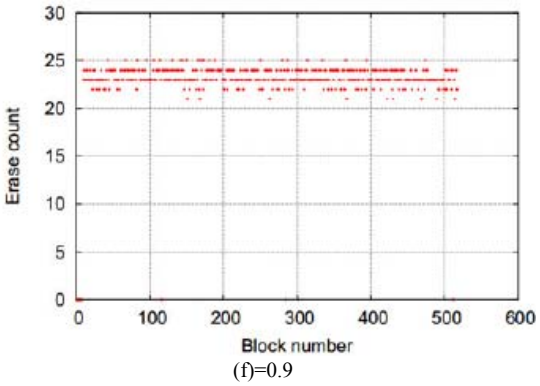
(a)=0.3



(a)=0.5



(d)=0.7



(f)=0.9

Fig. 7. Erase count distribution of all flash blocks.

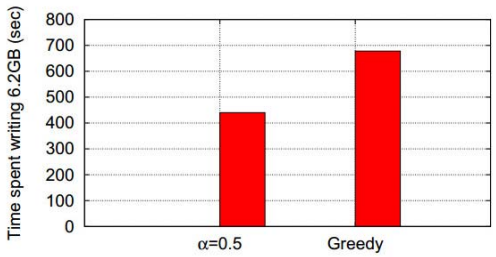


Fig. 8. 6.2GB burst write test.

Fig. 6 shows the average time spent in the garbage collection. As can be seen, our scheme shows low garbage collection overhead as greedy algorithm. It is because our several policies such victim block selection, hot/cold page classification, and free block management has low temporal overhead. Through these results, we can conclude that there is little difference in terms of efficiency between greedy algorithm and our scheme. In order to confirm this fact again, we evaluated them with the AS-SSD benchmark tool [12]. As can be seen in TABLE I, there is little difference between the proposed scheme and greedy algorithm.

Fig. 7 shows the erase count distribution of all blocks when different schemes are used. This distribution shows how the garbage collection schemes evenly wear out all blocks. As can be seen in Fig. 6 (a), when α is 0.1, the erase counts are almost uniformly distributed. As the value of α increases, the distribution of erase counts becomes spread out.

Through the previous results, we could find that an appropriate value of α is 0.3 or 0.5 for both efficiency and longevity. The reason is that α does not make a large effect on efficiency except that α is 0.1. On the other hand, when α is 0.3 or 0.5, the proposed scheme shows better performance in terms of longevity.

Finally, we performed another experiment to see the effectiveness of considering the SSD's internal architecture for garbage collection design. For this experiment, we made a simple workload that writes 6.2GB of data with a bursty random pattern. As can be seen in Fig. 8, our scheme shows lower write operation time than greedy algorithm by 66% when α is 0.5.

V. Conclusion

In this paper, we presented a new garbage collection scheme that can be practically used in SSDs. Our garbage collection scheme successfully considers both efficiency and longevity with low spatial and temporal overhead. In addition, we introduced a method that exploits the internal architecture of SSDs for garbage collection design. Through extensive experiments, we demonstrated that the proposed scheme exhibits good performance in terms of both efficiency and longevity.

References

[1] O. Kwon, K. Koh, J. Lee, and H. Bahn, "FeGC: An efficient garbage collection scheme for flash memory based storage systems," *Journal of Systems and Software*, vol. 84, no. 9, pp. 1507–1523, 2011.

[2] The OpenSSD project. [Online]. Available: http://www.openssd-project.org

[3] M. Wu and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," *in Proc. Architectural Support for Programming Languages and Operating Systems(ASPLOS)*, San Jose, CA, 1994, pp. 86-97.

[4] A. Kawaguchi, S. Nishioka, and H. Motoda, "A flash-memory based File System," *in Proc. USENIX Annual Technical Conference*, New Orleans, Louisiana, 1995, pp. 155-164.

[5] M. L. Chiang and R. C. Chang, "Cleaning policies in mobile computers using flash memory," *Journal of Systems and Software*, vol. 48, no. 3, pp. 213-231, 1999.

[6] H. J. Kim, S. G. Lee, "A new flash memory management for flash storage system," in *Proc. 23th Annual International Computer Software and Applications Conference*, Phoenix, Arizona, 1999, pp. 284-289.

[7] M. C. Wookey, "YAFFS Specification," 2001.

[8] D. Shmidt, "TrueFFS wear-leveling mechanism," Technical report, M-Systems, 2002.

[9] L. P. Chang, "On efficient wear leveling for large-scale flash-memory storage systems," in *Proc. ACM symposium on Applied computing (SAC '07)*, New York, 2007, pp. 1126-1130.

[10] R. Sykes, OCZ Technology, "Critical Role of Firmware and Flash Translation Layers in Solid Stat Drive Design," presented at the Flash Memory Summit, Santa Clara, CA, 2012.

[11] D. D. Levine, "Iometer User's Guide," Intel Server Architecture Lab, 2003.

[12] ALEX Intelligent Software. [Online]. Available: www. http://alex-is.de.

Taeho Nam is a student of the Department of Computer engineering, Kwangwoon University, Seoul, Korea. He is pursuing Bachelors in Computer engineering. His research interests include operating systems, flash memories and next-generation nonvolatile memories, and embedded system.

Taeseok Kim received the BS, MS and PhD degrees in computer science from Seoul National University, Korea, in 2000, 2002, and 2007 respectively. He iscurrently an assistant professor in the department of computer engineering, Kwangwoon University, Seoul, Korea. His research interests include multimedia systems, operating systems, storage systems, and embedded system.