An Efficient Garbage Collection for Flash Memory-Based Virtual Memory Systems

Seunggu Ji and Dongkun Shin, *Member*, IEEE

Abstract — *As more consumer electronics adopt monolithic kernels, NAND flash memory is used for the swap space in virtual memory systems. While flash memory has the advantages of low-power consumption, shock-resistance and non-volatility, it requires garbage collection due to its erase-before-write characteristic. The efficiency of the garbage collection scheme largely affects the performance of flash memory. This paper proposes a novel garbage collection technique which exploits data redundancy between the main memory and flash memory in flash memory-based virtual memory systems. Compared to the previous approach, our proposed scheme takes into consideration the locality of data to minimize the garbage collection overhead. In addition, by considering the computational overhead of the garbage collection algorithm, we also propose an adaptive scheme which can minimize the computational overhead with marginal I/O performance degradation. Experimental results demonstrate that the proposed garbage collection scheme improves performance by 37% on average compared to those of previous schemes*[1].

Index Terms — NAND flash memory, Flash Translation Layer (FTL), Garbage Collection, Virtual Memory, Buffer Management.

I. INTRODUCTION

NAND flash memory is widely used in constructing storage units for consumer electronics such as cellular phones, digital cameras and portable media players because of its merits of low power consumption, high random access performance and high shock-resistance. Flash memory is a good device for use as swap space in virtual memory system as well as for file and code storage due to its low access cost [1, 2, 3]. Compared with hard disk drives, flash memory can reduce the page swapping cost significantly. As more consumer electronics adopt monolithic kernels such as embedded Linux, flash memory-based virtual memory systems will become more popular. However, most research on flash memory has focused on flash file systems, with only a few studies on flash memory-based virtual memory systems.

The characteristics of flash memory are quite different from those of hard disk drives. A flash memory chip is composed of several blocks and each block consists of multiple pages. For example, in a large block multi-level-cell (MLC) NAND flash memory, one block is composed of 128 pages of 4 KB each. Flash memory supports the three commands of read, program (write) and erase. While the units for read and program commands are pages, the unit for erase command is a block. A flash memory page cannot be overwritten if it has already been programmed, and the corresponding block should be erased before data is written to the page. This feature is called the 'erase-before-write' constraint. Therefore, most flash storage systems write the updated data to other non-programmed pages, invalidating the old pages. This requires an address mapping scheme which translates the logical address used in the operating systems to the physical address used in the flash memory.

In order to handle these special features, a software layer called the flash translation layer (FTL) is usually used between the file system and the flash memory [4, 5, 6]. The FTL has two main functions. The first is address mapping, which can be divided into three categories depending on the granularity: block-level, page-level and hybrid mappings. The second function of FTL is garbage collection (GC) to reclaims the flash pages that have been invalidated by the update operations. The GC has three steps, i.e., victim block selection, valid page migration and victim block erase. The victim block selection identifies the victim block that will invoke the lowest GC cost, i.e., that with the smallest number of page migrations. The valid page migration moves the valid pages from the victim block to other clean blocks. The last step erases the victim block for future write requests.

The garbage collection invokes significant overhead since it requires a large number of page migrations and block erasures. Therefore, an efficient GC scheme is essential for high performance flash memory storage systems. There have been many studies on garbage collection in flash memory storage; however, only a few works have focused on the GC for flash memory-based virtual memory systems.

When flash memory is used as swap space, the GC should exploit the data redundancy between the main memory and the flash memory in order to eliminate unnecessary page copying. When a virtual memory page is swapped in, this page exists in both the main memory and the flash storage. Since this page will be written back to the flash memory when it gets swapped out at the next time, there is no need to copy these duplicated pages during GC, as is done in the duplication-aware garbage collection (DA-GC) [7]. DA-GC targets the page-level address mapping FTLs and thus shows a good performance at page-level mapping.

We found that DA-GC cannot display its merits for hybrid mapping FTLs without considering the locality information. For

[1] 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 (2010-0010387).

S. Ji and D. Shin are with the School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea (e-mail: dongkun@skku.edu).

Contributed Paper
Manuscript received 09/17/10
Current version published 12/23/10
Electronic version published 12/30/10.

this reason, we propose locality and duplication-aware garbage collection (LDA-GC) algorithms for flash memory-based virtual memory systems, consisting of the locality and duplication-aware victim block selection technique (LDA-VBS) and the locality and duplication-aware block merge technique (LDA-BM). These techniques significantly reduce the GC overhead in the hybrid mapping FTL by considering the update probability of duplicated data. The experiments using a trace-driven simulator show that the proposed techniques can improve the overall flash I/O performance, on average, by 37% compared to that of the existing duplication-unaware garbage collection (DU-GC) scheme for virtual memory benchmarks.

The remainder of the paper is organized as follows: Section 2 provides a survey of the relevant literature on flash memory management techniques. Section 3 presents the motivations of this paper. The detailed descriptions on LDA-VBS and LDA-BM techniques are provided in Section 4. Section 5 presents the performance evaluation results. Finally, Section 6 concludes the paper.

II. Related Works

Most previous studies on flash memory have focused on address mapping schemes. Block-level mapping [4] maintains the translation information between the logical block address and the physical block address; therefore, the offsets of a page are the same within both the logical block and the physical block. In page-level mapping [5], a logical page address is translated into a physical page address. Due to the independent management of pages, page-level mapping is more efficient than block-level mapping, but it requires a large memory space for the mapping table.

Hybrid mapping [6, 8, 9] uses both page-level mapping and block-level mapping and reserves a portion of the flash blocks as a log buffer. Hence, hybrid mapping FTLs are called the log buffer-based FTLs. Blocks in the log buffer are called the log blocks. The normal data blocks use block-level mapping, while the log blocks use page-level mapping. All write requests are first sent to the log buffer; if there is no free space in the log buffer, then valid data in a victim log block are moved into data blocks to make free space. The hybrid mapping FTL technique can yield high performance with a small mapping table. Therefore, most FTLs employ the hybrid mapping technique.

There are several studies on log buffer-based FTLs. The block-associative sector translation scheme (BAST) [6] associates a log block with only one data block; that is, when any page of a data block is updated, the new data should be written to the associated log block. The GC is invoked when there are no clean pages in the associated log block or when no log block is associated with the target data block; this occurs frequently for random writes. The GC selects one of the log blocks and moves all valid pages of the log block and its associated data block to a clean block. The log block and the data block are then erased and are exploited as new log blocks.

A drawback of the BAST scheme is its frequent GCs for random write patterns. To solve this problem, the fully-associative sector translation (FAST) scheme was proposed [8], in which one log block can be associated with multiple data blocks. Therefore, frequent GC invocations for random write requests can be prevented. However, the FAST scheme has a large GC cost once garbage collection is invoked because it moves many valid pages in several data blocks that are associated with the victim log block.

Generally, flash memory storage systems have a buffer cache to hide the long latency of flash memory. Buffer cache management is important for achieving high performance since the I/O requests on flash memory change depending on the buffer cache management technique. There are several flash-aware buffer management schemes, including FAB [10], CFLRU [11] and BPLRU [12]. However, these techniques do not consider duplicated pages. Lee *et al*. [13] have proposed a buffer-aware garbage collection (BA-GC) technique which exploits duplicated pages that are written in both the buffer cache and the flash memory. During garbage collection, the duplicated dirty pages are evicted into the flash memory to eliminate unnecessary page migrations.

Li *et al*. [7] proposed the duplication-aware garbage collection (DA-GC) technique for flash memory-based virtual memory systems. DA-GC does not move the duplicated pages in the flash memory during the valid page migration. Therefore, the pages are removed from the flash memory after GC erases the victim blocks; however, these pages remain in the main memory. Since the target of the DA-GC technique is the swap space of virtual memory systems, there is no critical consistency problem even when the duplicated pages are lost from the main memory due to a sudden power failure. The duplicated pages, which remain only in the main memory after GC, are written in the flash memory when they are swapped out. Although DA-GC can reduce the garbage collection overhead of the page-level mapping FTL, it may increase the garbage collection overheads of hybrid mapping FTLs because it generates more write requests on the log buffer and thus invokes frequent GCs in hybrid mapping. Our proposed techniques are based on the DA-GC scheme. However, our locality-aware approaches solve the problem of DA-GC in hybrid mapping FTLs.

III. Motivation

In this section, we introduce the DA-GC technique and its problem in hybrid mapping where the log blocks are used as the write buffer for data blocks. Fig. 1 shows an example of duplication-*unaware* garbage collection (DU-GC) in the FAST hybrid mapping. The page cache has six pages that are sorted by their access recencies. Page P2 is the least-recently-used (LRU) page, and page P9 is the most-recently-used (MRU) page. The pages P2 and P11 are dirty (i.e., the page cache and the flash memory have different data), and the remaining pages are clean. Flash memory consists of seven physical blocks whose physical block numbers (PBNs) are 0～6; PBN 0, PBN 1 and PBN 2 are allocated for data blocks, and PBN 3 and PBN 4 are allocated for log blocks. Since the data blocks are managed using block-level mapping, all pages are written at the specified page offsets within the data block.

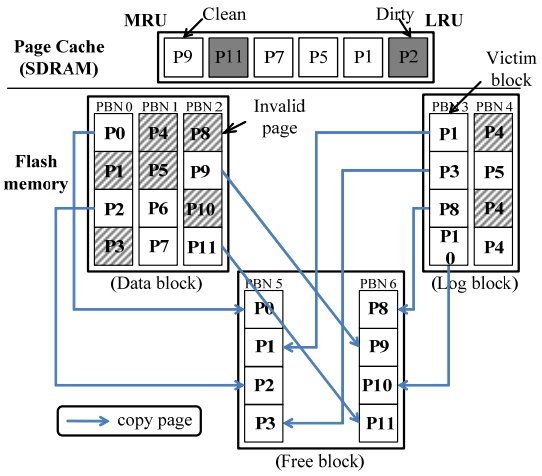

Fig. 1 An example of duplication-unaware garbage collection.

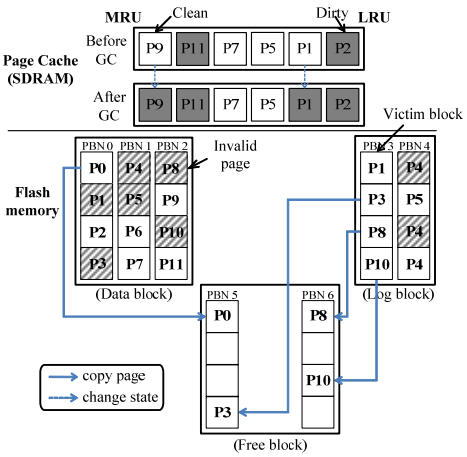

Fig. 2 An example of duplication-aware garbage collection.

PBN 5 and PBN 6 are free blocks reserved for garbage collection. We assume that each flash block is composed of four pages. For the sequence of write requests on the logical pages, (P1, P3, P8, P10, P4, P5, P4, P4), the log blocks contain these pages, and the corresponding pages in the data blocks are invalidated. Each log block can be associated with multiple data blocks. For example, PBN 3 in the log buffer is associated with data blocks PBN 0 and PBN 2, and PBN 4 is associated with only PBN 1.

Garbage collection should be invoked when there is no free space in the log buffer. If PBN 3 is selected for a victim block, then the GC copies all valid pages in PBN 3 and its associated data blocks (PBN 0 and PBN 2) into the free blocks, PBN 5 and PBN 6. The valid pages P0□P3 and P8□P11 are copied into PBN 5 and PBN 6, respectively. Since the victim log block and its associated data blocks are merged into free blocks, this step is called the block merge. After the block merge operation is completed, PBN 5 and PBN 6 are changed into data blocks, and PBN 0, PBN 2 and PBN 3 are erased, with one of them being allocated as a new log block.

The DU-GC does not consider the duplicated pages in the page cache. If we have information on the page cache, a more efficient GC can be implemented by considering the duplicated pages in both the page cache and the flash memory. For many embedded devices such as mobile phones, the page cache and the FTL can share their information because both of them are executed on the same processor. If the page cache manager can notify the FTL of the dirty page information, unnecessary page migrations can be prevented.

Fig. 2 shows the duplication-aware garbage collection (DA-GC) scheme [7]. When GC selects PBN 3 as a victim block, it does not copy the duplicated pages P1, P2, P9, and P11 since they are also contained in the page cache. Therefore, the number of page migrations is reduced by half. Instead, P1 and P9 are changed into dirty states in the page cache since they should be written in the flash memory when they are evicted from the page cache.

Although the DA-GC scheme significantly reduces the block merge cost, more pages will be sent to the flash memory from the page cache since all of the duplicated clean pages are changed into dirty pages. The increased number of page evictions in the DA-GC scheme has no adverse effects on garbage collection in page-level mapping since the pages can be written at any location in a block. Therefore, DA-GC is an effective technique in page-level mapping. However, DA-GC may invoke frequent garbage collections with hybrid mapping. For instance, as shown in Fig. 2, four physical pages in PBN 5 and PBN 6 are not utilized in DA-GC. Instead, when pages P1 and P2 are evicted from the page cache due to page replacement, they should be written in the log blocks. As a result, the log blocks more quickly consume free space.

If these pages remain clean until they are evicted from the page cache, i.e., garbage collections or host requests do not make these pages dirty, then they will not be written in the flash memory. In particular, since page P1 has not recently been used, there is little possibility for the page to be updated (and consequently changed to dirty) before it is evicted from the page cache. Therefore, it is better to copy page P1 during the block merge and leave it clean. However, since page P9 is the MRU page, it is likely to become dirty even though DA-GC does not change its state. Therefore, even if page P9 is excluded from page migrations by DA-GC, there may be no benefit with regard to the GC cost. Consequently, the duplication-aware scheme should be applied selectively considering the localities of the duplicated pages.

To solve the problem of DA-GC in hybrid mapping, we propose a locality-aware victim block selection technique, called LDA-VBS, and a locality-aware block merge technique, called LDA-BM, for DA-GC. These techniques divide the page cache into two regions, LRU and MRU regions, and use different policies for each region. The LDA-VBS technique selects the victim log block that invokes a small number of state changes for the duplicated clean pages in the LRU region of the page cache. The LDA-BM technique determines whether to copy each duplicated page during a block merge based on the locality of the corresponding page in the page cache. In addition, we propose the LRU dirty page

eviction (LDE) technique that forces dirty pages in the LRU region of the page cache to be evicted during garbage collection in order to reduce unnecessary page migrations. The proposed techniques can prevent frequent garbage collections during hybrid mapping while exploiting the advantage of DA-GC that reduces unnecessary copies of duplicated pages.

IV. LOCALITY-AWARE GARBAGE COLLECTION

A. Locality and Duplication-Aware Victim Block Selection

General victim block selection algorithms consider only the block merge cost when selecting a victim block. However, in order to prevent duplicated clean pages in the LRU region of the page cache from being changed into dirty, we should consider not only the merge cost but also the potential loss resulting from the increase in write requests in the DA-GC. The proposed LDA-VBS technique optimizes both the garbage collection overhead and the potential loss.

Under the DA-GC scheme, we can represent the garbage collection overhead, $C_{GC}(L_i)$, for a victim log block, L_i, as follows:

$$C_{GC}(L_i) = (A(L_i)+1)\cdot C_e + \delta(L_i)\cdot(C_r + C_w), \quad (1)$$

where $A(L_i)$ and $\delta(L_i)$ denote the number of data blocks associated with L_i and the number of non-duplicated (i.e., they exist only in the flash memory) valid pages in L_i or its associated data blocks, respectively. For example, in Fig. 2, A(PBN 3) is 2 and δ(PBN 3) is 4. C_e, C_w and C_r represent the timing costs for block erase, page write and page read in the flash memory, respectively. Only $\delta(L_i)$ number of flash page reads and writes are required since DA-GC does not copy the duplicated pages during the block merge. After the block merge is completed, $A(L_i)$ number of data blocks and one log block are erased. Therefore, $A(L_i)$ + 1 number of block erases are required.

However, as explained in Section 3, DA-GC changes the duplicated clean pages in the page cache into dirty pages, invoking more write requests from the page cache to the flash memory. Therefore, DA-GC has potential loss as follows:

$$C_{loss}(L_i) = \gamma(L_i)\cdot(C_w + \alpha), \quad (2)$$

where $\gamma(L_i)$ represents the number of duplicated pages of the log block L_i whose corresponding pages in the page cache are changed from clean into dirty by DA-GC and are not updated further by following host requests until they are evicted. In Fig. 2, two clean pages, P9 and P1, are changed into dirty by DA-GC. However, since the MRU page P9 has a high possibility of being changed to dirty by host requests, the value of $\gamma(L_i)$ will be less than 2. The cost for writing the dirty pages into the flash memory is $\gamma(L_i)$ C_w. In addition, the write requests invoke more garbage collections. We add the overhead cost of α that represents the average block merge cost per one dirty page write. The approximate value of α is $C_r+C_w+C_e/N_{page}$ because a dirty page written in the flash memory invokes one page read/write for page migration and one block erase per N_{page} number of pages, where N_{page} represents the total number of flash pages in a flash block.

However, it is impossible to know the exact values of $\gamma(L_i)$ during GC without knowledge of future host requests. To predict these values, we used the 3-region LRU cache [13] in which a page cache is divided into three regions: an MRU region, an LRU region, and an initial region, each of which provides the update probability of a page in the region. By dynamically adjusting the size of each region based on the transition rates between the three regions, the 3-region LRU buffer identifies the page cache access pattern. The update probability of each duplicated page in the page cache can be determined based on the region information including the page.

To consider both the garbage collection overhead and the potential loss, the overall garbage collection cost can be represented as follows:

$$C_{total}(L_i) = C_{GC}(L_i) + C_{loss}(L_i) \quad (3)$$

The LDA-VBS technique selects the victim block with the lowest value of $C_{total}(L_i)$ in order to prevent DA-GC from invoking a large potential loss.

B. Locality and Duplication-Aware Block Merge

Since the pages in the page cache have different probabilities of being updated, the LDA-BM technique uses different policies depending on the future access probability of each page. The clean pages in the MRU region of the page cache have high probabilities of being changed to dirty before they are evicted from the page cache even though DA-GC does not change their states. On the contrary, the clean pages in the LRU region of the page cache have low possibilities of being changed to dirty by host requests. Therefore, it may be beneficial not to apply the DA-GC technique to the clean pages in the LRU region, i.e., not to change the clean data in the LRU region into dirty data. Then, the page migration cost of GC increases compared to that of DA-GC. However, we can reduce the frequency of GCs that may invoke large overhead. (The initial region of the 3-region cache is regarded as being included in the MRU region for simple implementation.)

Fig. 3 shows the proposed LDA-BM technique. Page P1 in the flash memory is copied during the block merge operation and P1 in the page cache remains clean since the page is in the LRU region of the page cache. However, the clean page in the MRU region P9 and the dirty pages P2 and P11 are not copied during the block merge. Even though page P9 is changed to dirty, the potential loss due to the change will be small since it has a high possibility of being changed to dirty by future host requests.

For the LDA-BM technique, the victim block selection policy should be modified. We can represent the garbage collection overhead, $C_{GC}(L_i)$, for a victim log block, L_i, as follows.

$$C_{GC}(L_i) = (A(L_i)+1)\cdot C_e + (\delta(L_i)+\pi(L_i))\cdot(C_r + C_w), \quad (4)$$

where $\pi(L_i)$ denotes the number of duplicated pages of the log block L_i whose corresponding pages in the page cache are duplicated clean pages in the LRU region. Compared to the DA-GC cost in Equation (1), LDA-BM has a larger GC cost since it requires more flash page migrations. However, LDA-BM reduces the potential loss of DA-GC since it does not change

the clean pages in the LRU region into dirty pages. Therefore, the potential loss of LDA-BM is as follows:

$$C_{loss}(L_i) = \gamma_{MRU}(L_i) \cdot (C_w + \alpha), \quad (5)$$

where $\gamma_{MRU}(L_i)$ is the number of duplicated pages of the log block L_i whose corresponding pages in the page cache are clean pages in the MRU region and are changed from clean into dirty by DA-GC. The potential loss is smaller than that in Equation (2) since $\gamma(L_i)$ is larger than $\gamma_{MRU}(L_i)$.

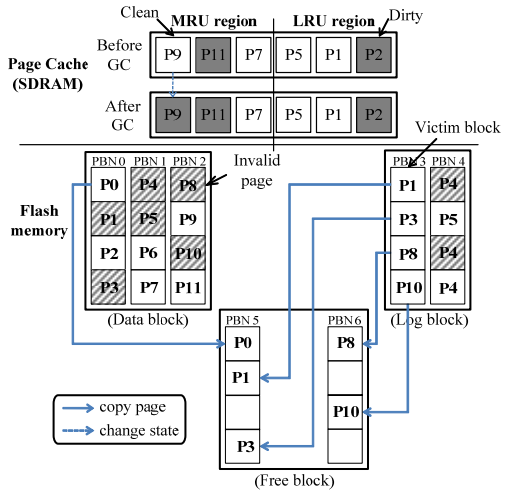

Fig. 3 An example of LDA-BM.

C. LRU Dirty Page Eviction

The LRU dirty page eviction (LDE) technique exploits the duplicated dirty data in the LRU region of the page cache in order to reduce the GC cost. It is better to move the duplicated dirty pages in the LRU region of the page cache into the flash memory during garbage collection because the pages have high probabilities of being evicted to the flash memory without further updates. Then, we can efficiently utilize the data blocks by reducing the number of flash memory spaces unutilized by DA-GC. The copied dirty pages in the page cache are changed to clean. For example, if we use the LDE technique for the case in Fig. 3, page P2 is copied from the page cache to the flash block PBN 5 and is changed into clean in the page cache. Then, when page P2 is evicted from the page cache, there is no write request to the log block of flash memory.

We can simultaneously use both LDA-BM and LDE techniques during the block merge operation to apply different policies for the duplicated pages in the LRU region of the page cache. While LDA-BM is applied to the duplicated clean pages, LDE is applied to the duplicated dirty pages. We can reduce the potential garbage collection overhead invoked by DA-GC by using two techniques in the LRU region of the page cache.

When both the LDA-BM and LDE techniques are used, the garbage collection overhead, $C_{GC}(L_i)$, for a victim log block, L_i, is calculated as follows:

$$C_{GC}(L_i) = (A(L_i)+1) \cdot C_e + (\delta(L_i) + \pi(L_i)) \cdot (C_r + C_w) + \theta(L_i) \cdot (C_b + C_w) \quad (6)$$

where $\theta(L_i)$ denotes the number of duplicated pages of the log block L_i whose corresponding pages in the page cache are duplicated dirty pages in the LRU region, and C_b represents the transfer cost of a page from the page cache to the flash memory. We assume that C_b is larger than C_r due to the bus transaction. Compared to the DA-GC cost in Equations (1) and (4), using both LDA-BM and LDE techniques invokes a larger GC cost since $\theta(L_i)$ number of pages should be copied from the page cache to the flash memory. However, the LDE technique has a potential benefit. Since LDE changes the duplicated dirty pages in the LRU region of the page cache into clean pages, the number of write requests to the log blocks is reduced. Therefore, the total GC cost is as follows:

$$C_{total}(L_i) = C_{GC}(L_i) + C_{loss}(L_i) - C_{benefit}(L_i), \quad where \quad C_{loss}(L_i) = \gamma_{MRU}(L_i) \cdot (C_w + \alpha) \text{ and } C_{benefit}(L_i) = \gamma_{LRU}(L_i) \cdot (C_w + \alpha). \quad (7)$$

In this equation, $\gamma_{LRU}(L_i)$ represents the number of duplicated pages of the log block L_i whose corresponding pages in the page cache are dirty pages in the LRU region and are changed into clean by LDE without being updated by following host requests.

TABLE I
THE STATE CHANGES OF A DUPLICATED PAGE UNDER DIFFERENT SCHEMES.

page cache area	state before GC	count	state after GC		
			DA-GC LDA-VBS	LDA-BM	LDA-BM /LDE
MRU region	Dirty	$\rho(L_i)$	Dirty	Dirty	Dirty
	Clean		Dirty	Dirty	Dirty
LRU region	Dirty	$\theta(L_i)$	Dirty	Dirty	Clean
	Clean	$\pi(L_i)$	Dirty	Clean	Clean

Table I summarizes the state changes of a duplicated page under each scheme. DA-GC and LDA-VBS have the lowest GC costs but have the largest potential losses on future GC costs since they change all of the duplicated clean pages in the page cache into dirty pages. LDA-BM has a higher GC cost than does DA-GC but has a lower potential loss because it does not change the states of the duplicated clean pages in the LRU region. Using LDE in addition to LDA-BM, the GC cost increases; however, there is a potential benefit since the duplicated dirty pages in the LRU region are changed into clean pages.

D. Adaptive LDA-VBS

Even though the proposed LDA-VBS technique can significantly reduce the flash memory I/O cost, it invokes high computational overhead to identify the duplicated pages at every garbage collection. The victim block selection algorithm should determine whether each page in a log block has a duplicated page in the page cache. In particular, the computational overhead increases in proportion to the number of log blocks in a log buffer. Since the garbage collection overhead is generally reduced as the size of the log buffer increases, high-performance flash storage systems prefer a large log space. Consequently, the high complexity of LDA-VBS could be a burden for such systems.

To overcome this problem, only a portion of the log buffer can be examined when choosing a victim log block. Our approach is to use a victim window which includes k number of

log blocks close to the LRU position in a log buffer and inspect only the log blocks within the victim window, instead of examining all log blocks. The log blocks in the LRU position tend to have a relatively low block merge cost because they are likely to have a small number of valid pages. Therefore, this approach can reduce the computational overhead without significant damage to the performance of LDA-VBS.

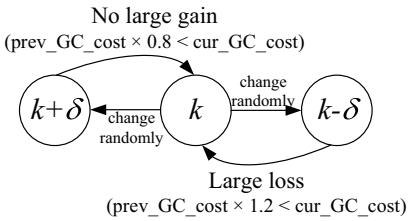

Fig. 4 Victim window adaptation.

There is a trade-off between the I/O performance and the computational overhead, i.e., with a large victim window, a better victim log block can be selected but the computational overhead is significantly higher. Therefore, it is important to choose a proper victim window size k during the victim selection step. Since the optimal value of k depends on the workload pattern, we adjust it by observing the garbage collection cost. As shown in Fig. 4, we change the victim window size k into $k + \delta$ or $k - \delta$ and then observe the change in GC cost. If the current GC cost (cur_GC_cost) with the victim window size $k + \delta$ is not reduced by more than 20% compared to the previous GC cost (prev_GC_cost), we restore the victim window size into k. On the other hand, if the current GC cost with the victim window size $k - \delta$ is increased by more than 20% compared to the previous GC cost, the victim window size is restored into k. Using this adaptation algorithm, the smallest victim window size which has only a small difference in computational overhead compared to the unlimited victim window size can be determined.

V. EXPERIMENTS

A. Experimental Environments

We implemented a trace-driven simulator in order to evaluate the performances of the proposed schemes. The simulator consists of the page cache simulator and the storage simulator. The page cache is managed by the 3-region LRU algorithm [13] to divide it into the LRU and MRU regions. We used five real virtual memory traces collected using Valgrind toolset, which are captured while executing several applications, acrobat, gqview, kword, mozilla and office, on a Linux system. The flash memory model used in the simulation was based on Samsung SLC large block NAND flash memory [14], in which each flash block is composed of 64 pages and each page is 2 KB. The timing delays of page read, page write and block erase (C_r, C_w and C_e) are 25 usec, 200 usec and 2 msec, respectively.

The seven schemes, shown in Table II, were compared. Each scheme used different victim block selection and block merge techniques. All schemes used the FAST hybrid mapping FTL [8]. We assumed that the normal VBS algorithm was the round-robin (RR) selection policy, which selects the oldest log block as the victim. Since the oldest block generally has a small number of valid pages, the RR policy invokes a small GC cost and thus it is a reasonable solution.

TABLE II A SUMMARY OF THE EVALUATED SCHEMES.

schemes	victim block selection	block merge	LDE
DU-GC	RR	DU-BM	No
DA-GC	RR	DA-BM	No
LDA-GC$_1$	LDA-VBS	DA-BM	No
LDA-GC$_2$	RR	LDA-BM	No
LDA-GC$_3$	LDA-VBS	LDA-BM	No
LDA-GC$_4$	RR	LDA-BM	Use
LDA-GC$_5$	LDA-VBS	LDA-BM	Use

RR : Round-Robin policy

B. Performance Comparison

Fig. 5 presents the total I/O execution times of the examined GC schemes normalized by those of DU-GC. The I/O execution times include the flash read, write and erase costs invoked by the garbage collection as well as the page swap-out. The page cache size is 4 MB and the flash memory has 32 log blocks. The performance of DA-GC was similar or inferior to that of DU-GC because the potential loss of DA-GC was larger than the GC cost reduction resulting from not copying the duplicated pages. By comparing the results of LDA-GC$_1$ and LDA-GC$_2$, it can be known that the LDA-VBS technique (which has a performance improvement of 10% on average) is more effective than is the LDA-BM technique (which has a performance improvement of 6% on average) because LDA-VBS significantly reduces the garbage collection overhead as well as the potential loss.

The LDA-GC$_3$ scheme, which uses both LDA-VBS and LDA-BM, showed more significant performance improvements (by 24% on average) due to the synergetic effect of the two techniques. The LDA-GC$_4$ scheme, which uses both LDA-BM and LDE, improved the performance by 28% on average. The LDA-GC$_5$ scheme, which uses all of the proposed techniques, reduced the I/O execution times by 37% on average compared to that of DU-GC.

To analyze the performance differences, we observed the behaviors of each garbage collection scheme. Fig. 6 shows the number of dirty page evictions from the page cache under each GC scheme normalized to that of DU-GC. Since the DA-GC scheme changes the duplicated clean pages of the page cache into dirty pages, it increases the number of page writes by 27% on average. The LDA-GC$_1$ scheme invokes a smaller number of page evictions since it selects the victim block with a low potential loss resulting from the increase in page evictions. However, it also invokes more page evictions compared to those of DU-GC. Using the DA-BM technique, the dirty page evictions were reduced to a similar level of that of DU-GA, as shown in the results of LDA-GC$_2$ and LDA-GC$_3$. The LDA-GC$_4$ and LDA-GC$_5$ schemes using both LDA-BM and LDE techniques showed smaller numbers of page evictions by 14□18% compared to that of DU-GC.

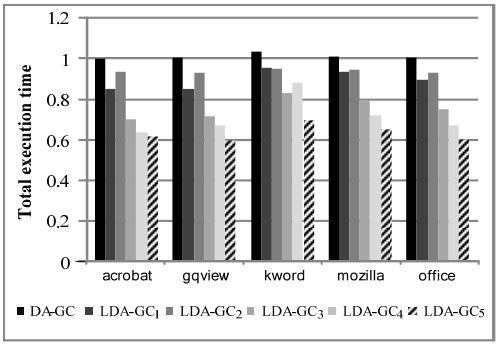

Fig. 5 Total I/O execution times normalized to that of DU-GC.

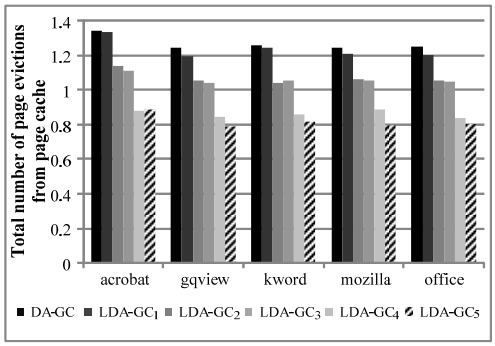

Fig. 6 Total number of page evictions from the page cache (normalized to that of DU-GC).

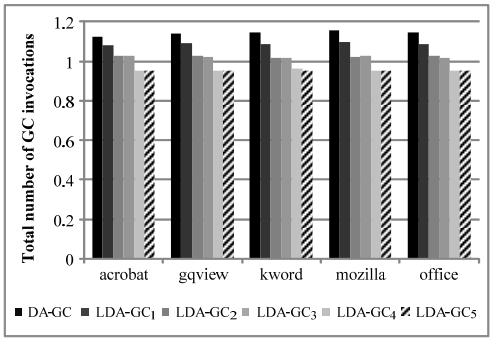

Fig. 7 Total number of GC invocations (normalized to that of DU-GC).

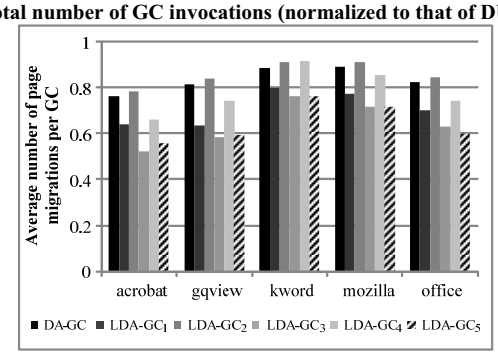

Fig. 8 Average number of page migrations per garbage collection (normalized to that of DU-GC).

The increased number of page evictions invokes frequent garbage collections for the log buffer of the flash memory. Fig. 7 shows the number of garbage collections invoked during benchmark executions under the proposed GC schemes. These values were normalized to those of DU-GC. While DA-GC, LDA-GC_1, LDA-GC_2 and LDA-GC_3 invoked larger numbers of GCs than did DU-GC due to the increased number of page evictions, LDA-GC_4 and LDA-GC_5 that used the LDE technique outperformed DU-GC by about 4～5% due to the potential benefit shown in Equation (7). However, since all of the proposed LDA-GC schemes showed performance improvements over DU-GC, according to the results in Fig. 5, it is inferred that the proposed LDA-GC schemes require smaller costs per GC invocation than does DU-GC.

Fig. 8 shows the number of page migrations during block merge operations. The DA-GC scheme required a smaller number of page migrations compared to that of the DU-GC scheme since it does not copy the duplicated pages during block merge operations. Since the LDA-VBS technique selects the victim log block considering the page migration cost, LDA-GC_1, LDA-GC_3, and LDA-GC_5 showed smaller numbers of page migrations compared to those of DA-GC, LDA-GC_2, and LDA-GC_4, respectively. The number of page migrations was increased slightly by the LDA-BM technique (as shown in the results of LDA-GC_2) since the technique copies the duplicated pages whose corresponding pages in the page cache are clean pages in the LRU region.

The LDE technique copies the dirty duplicated pages in the LRU region of the page cache into the flash memory during block merge operations. Therefore, we can expect that the page migrations will be increased further by the LDE technique. However, the LDA-GC_4 scheme, which uses both the LDA-BM and LDE techniques, showed a smaller number of page migrations compared to that of DA-GC. This is because the LDE technique reduced the number of full merge operations, as shown in Table IV, which shows the number of block merges according to their type, normalized with respect to that of DU-GC. The merge cost of full merge is higher than those of switch merge and partial merge because full merge requires a large number of erase operations and page migrations [6]. Therefore, it is important to reduce the number of full merges in order to minimize the garbage collection cost.

While DA-GC increased the numbers of all types of merges, LDA-GC_3, LDA-GC_4 and LDA-GC_5 reduced the number of full merge operations because they mitigated the randomness of write requests on the log buffer by reducing dirty page evictions from the page cache. Fig. 9 shows the average number of erase operations per garbage collection. The GC schemes invoking fewer full merge operations generated fewer erase operations. From the results of Fig. 7 and Fig. 9, we show that the LDA-GC_3, LDA-GC_4 and LDA-GC_5 schemes can prolong the lifespan of flash memory which has the program/erase cycle limit since they consume smaller numbers of program/erase cycles per garbage collection and invoke smaller numbers of garbage collections.

The reduced page migrations and erase operations of the proposed techniques affect the average cost per garbage collection, as shown in Fig. 10. All GC schemes provided smaller average GC costs than did DU-GC. Since the LDA-GC_2 scheme invoked more page migrations, as shown in Fig. 8, and less erase operations, as shown in Fig. 9, it has a similar GC cost to that of DA-GC. The average GC costs of LDA-GC_1, LDA-GC_3, and LDA-GC_5, which use the LDA-VBS technique, were lower than those of other schemes since LDA-VBS considers the GC cost of the victim log block. The LDA-GC_5 scheme achieved the best performance since it had the lowest average GC cost, as shown in Fig. 10, and the smallest number of GC invocations, as shown in Fig. 7.

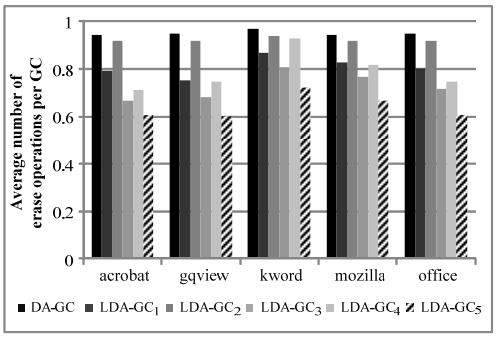

Fig. 9 Average number of erase operations per one garbage collection (normalized to that DU-GC).

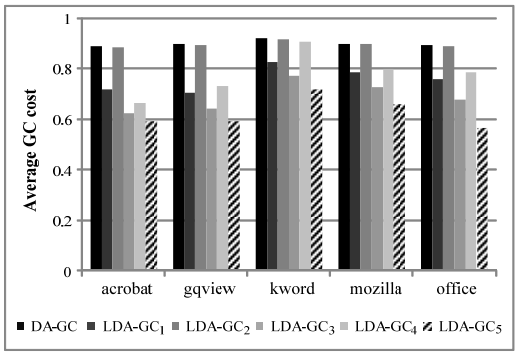

Fig. 10 Average garbage collection cost (normalized to that DU-GC).

C. *The Effects of Page Cache Size and Log Buffer Size*

We also evaluated the effect of the page cache size. Fig. 11 illustrates the total I/O execution time and the average number of duplicated pages excluded from page migrations (N_{dup}) for kword workload while varying the page cache size from 1 MB to 16 MB. The number of log blocks was fixed at 32. The performances improved as the page cache size increased because the hit ratio of the page cache increased. LDA-GC_3 and LDA-GC_5 showed better performances than did DU-GC, regardless of the page cache size. Moreover, as the page cache size increased, the performance gaps between the DU-GC and LDA-GC schemes increased since the number of duplicated pages increased. When there are a large number of duplicated pages, the proposed schemes have more chances to reduce the garbage collection overhead.

We also evaluated the effect of flash log buffer size. Fig. 12 shows the I/O execution time and the average number of duplicated pages while varying the number of log blocks in the flash memory from 8 to 128. The page cache size was fixed to 4 MB. As the number of log blocks increased, the execution times were reduced. When there are many log blocks, a long time is required for a log block to be selected as a victim block. Therefore, when a log block is selected as a victim block by the garbage collection, most of the pages in the victim block may be invalid, and thus the GC invokes a small page migration cost. The performance gaps between DU-GC and LDA-GC increased as the number of log blocks increased. This is because LDA-VBS can identify a better victim block when there are many log blocks available.

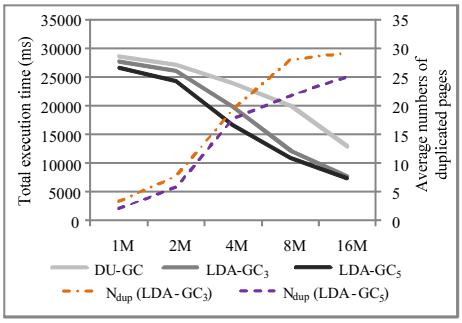

Fig. 11 I/O execution times and average numbers of duplicated pages when varying the page cache sizes (kword workload).

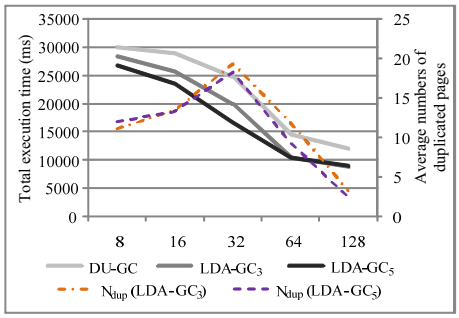

Fig. 12 I/O execution times and average numbers of duplicated pages when varying the number of log blocks (kword workload).

The average number of duplicated pages increased as the number of log blocks increased since LDA-VBS, which selects the log block with many duplicated pages, has more victim candidates. However, the value reached its peak when the number of log blocks was 32. If there are too many log blocks, the victim block has a small number of valid pages, and thus the number of duplicated pages decreases.

D. *Adaptive Victim Block Selection*

Fig. 13 shows the performance changes in LDA-GC_5 while varying the log block victim window for victim block selection. As the size of the victim window increased, performance improved since the LDA-VBS can select a better victim log block among more candidates. However, there were marginal changes in performance when the victim window was large (32～128). The computational overhead for finding duplicated pages decreases but the flash memory I/O cost increases as the victim window decreases. Therefore, it is important to select the smallest victim window in which the performance is not significantly degraded compared with that of the largest victim window. For example, the

optimal victim window size for kword trace is 64 since there is no large difference in performance when the victim window is larger than 64.

The adaptive LDA-VBS, explained in Section 4.4, can determine the optimal points, as shown in Fig. 14. The two LDA-VBS schemes using static victim windows of one page and 128 pages, respectively, were compared with the adaptive LDA-VBS using dynamic victim windows. We used two different values for the initial victim window size in the adaptive LDA-VBS scheme. The adaptive LDA-VBS schemes adjust their victim window sizes by observing the garbage collection cost. As a result, their performances were similar to that of the static scheme with the largest victim window.

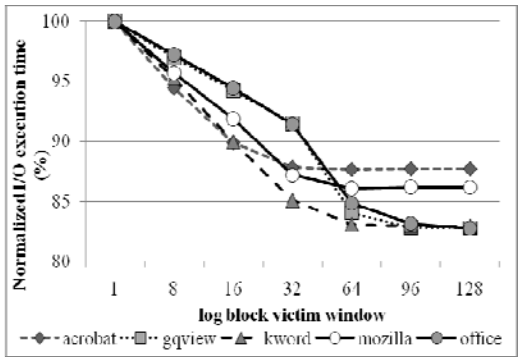

Fig. 13 I/O execution time comparison when varying the victim windows size.

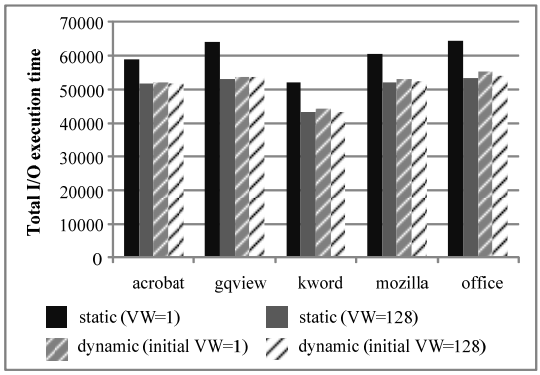

Fig. 14 I/O performance of the adaptive VBS scheme.

VI. CONCLUSIONS

Flash memory is a good device for use as swap space in virtual memory systems. For flash memory-based virtual memory systems, locality and duplication-aware garbage collection techniques are proposed to reduce garbage collection overhead by removing the duplicated pages from the flash memory. In order to solve the potential loss problem of the previous duplication-aware garbage collection technique in hybrid mapping FTLs, the proposed LDA-VBS technique considers both the garbage collection overhead and the potential loss. The LDA-BM and LRU dirty page eviction techniques selectively apply duplication-aware page migration depending on the locality of each page in the page cache. The experimental results showed that there was 37% of average improvement compared to that of DU-GC.

REFERENCES

[1] C. Park, J.-U. Kang, S.-Y. Park, and J.-S. Kim. Energy-aware demand paging on NAND flash-based embedded storages. In Proc. of ISLPED'04, pages 338–343, 2004.

[2] Y. Joo, Y. Choi, C. Park, S. W. Chung, E. Chung, and N. Chang. Demand paging for OneNAND flash eXecute-in-place. In Proc. of CODES+ISSS'06, pages 229–234, 2006.

[3] J. In, I. Shin, and H. Kim. SWL: a searchwhile-load demand paging scheme with NAND flash memory. In Proc. of LCTES'07, pages 217– 226, 2007.

[4] A. Ban. Flash file system optimized for page-mode flash technologies. United State Patent, No. 5,937,425, 1999.

[5] A. Ban. Flash file system. United State Patent, No. 5,404,485, 1995.

[6] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho. A space-efficient flash translation layer for compact flash systems. IEEE Trans. on Consumer Electronics, 48(2):366–375, 2002.

[7] H.-L. Li, C.-L. Yang, and H.-W. Tseng. Energy-aware flash memory management in virtual memory system. IEEE Trans. VLSI, 16(8):952–964, 2008.

[8] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. ACM Trans. on Embedded Computing Systems, 6(3), 2007.

[9] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J. S. Kim. A reconfigurable FTL (flash translation layer) architecture for NAND flash-based applications. ACM Trans. on Embedded Computing Systems, 7(4):1– 23, 2008.

[10] H. Jo, J. Kang, S. Park, J. Kim, and J. Lee. FAB: Flash-aware buffer management policy for portable media players. IEEE Trans. on Consumer Electronics, 52(2):485–493, 2006.

[11] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee. CFLRU: a replacement algorithm for flash memory. In Proc. of CASES'06, pages 234–241, 2006.

[12] H. Kim and S. Ahn. BPLRU: a buffer management scheme for improving random writes in flash storage. In Proc. of FAST'08, pages 1–14, 2008.

[13] S. Lee, D. Shin, and J. Kim. Buffer-aware garbage collection techniques for NAND flash memory-based storage systems. In Proc. of IWSSPS'08, pages 27–32, 2008.

[14] Samsung Electronics. 1G x 8 Bit / 2G x 8 Bit / 4G x 8 Bit NAND Flash Memory. http://www.samsung.com/global/business/semiconductor/products/flash/Products NANDFlash.html, 2007.

BIOGRAPHIES

Seunggu Ji received the B.S. degree in computer science from Dankook University, Korea in 2007. He is currently a Master student in the School of Information and Communication Engineering, Sungkyunkwan University. His research interests include embedded software, file systems and flash memory.

Dongkun Shin (M'08) received the B.S. degree, the M.S. degree, and the Ph.D. degree in computer science and engineering from Seoul National University, Korea, in 1994, 2000 and 2004, respectively. He is currently an Assistant Professor in the School of Information and Communication 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 multimedia and real-time systems.