2010 10th IEEE International Conference on Computer and Information Technology (CIT 2010)

Locality and Duplication-Aware Garbage Collection for Flash Memory-Based Virtual Memory Systems

Seunggu Ji and Dongkun Shin
School of ICE
Sungkyunkwan University
Suwon, Korea
{skyjsg, dongkun}@skku.edu

Abstract—**As embedded systems adopt monolithic kernels, NAND flash memory is used for swap space of virtual memory systems. While flash memory has the advantages of low-power consumption, shock-resistance and non-volatility, it requires garbage collections due to its erase-before-write characteristic. The efficiency of 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. The proposed scheme takes the locality of data into consideration to minimize the garbage collection overhead. Experimental results demonstrate that the proposed garbage collection scheme improves performance by 37% on average compared to previous schemes.**

Keywords-NAND flash memory, Flash Translation Layer (FTL), Garbage Collection, Virtual Memory, Buffer Management

I. INTRODUCTION

NAND flash memory is widely used in building storage systems of embedded systems such as cellular phones, digital cameras and MP3 players. NAND flash memory has the merits of low power consumption, high random access performance and high shock-resistance. Due to the price reduction of NAND flash memory, solid-state disk (SSD), which is composed of several flash chips, has recently been replacing hard disk drives (HDD) in desktop PCs and enterprise servers.

Flash memory is a good device for swap space of virtual memory systems as well as file and code storages due to its low access cost [1, 2, 3]. Compared with hard disk drive, flash memory can reduce the page swapping cost significantly. As embedded systems adopt monolithic kernels such as embedded Linux and MS Windows CE, flash memory-based virtual memory systems will be popular. However, most of researches on flash memory focused on the flash file systems and there are only a few studies on the flash memory-based virtual memory systems.

The characteristics of flash memory are quite different from those of hard disk. A flash memory chip is composed of several blocks and each block consists of multiple pages. For example, in a large block MLC NAND flash memory, one block is composed of 128 pages, and the size of each page is 4 KB. Flash memory supports three commands: read, program and erase. While the units for read and program commands are page, blocks are the units for erase command. We cannot overwrite new data at flash memory pages that have already been programmed. The corresponding block should be erased before data is written to the page. This feature is called 'erase-before-write' constraint. Therefore, most flash storage systems write the updated data to other non-programmed pages, and they invalidate the old page. This requires an address mapping scheme to translate a logical address used in operating systems to a physical address used in flash memory.

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

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

When flash memory is used for a swap space, the GC should exploit the data redundancy between the main memory and flash memory to eliminate unnecessary valid page copying. When a virtual memory page is swapped in, this page exists in both the main memory and flash storage. Since this page will be written back to flash memory when it gets swapped out next time, we can reduce page copying during GC if we do not move these duplicated pages of a victim block to free space. The duplication-aware garbage collection scheme (DA-GC) [6] uses the technique. DA-GC is designed assuming that FTL uses page-level address mapping and thus shows a good performance in the page-level mapping. However, DA-GC cannot improve the performance of garbage collection in the hybrid mapping.

In this paper, we propose the locality and duplication-aware victim block selection technique (LDA-VBS) and the locality and duplication-aware block merge technique (LDA-BM) for flash memory-based virtual memory systems. These techniques significantly reduce the DA-GC overhead in the hybrid mapping FTL by considering the update probability of duplicated data. We evaluated the proposed techniques using a trace-driven simulator. The experimental results show that the proposed techniques can improve the overall flash I/O performance, on average, by 37% compared to 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 at Section 4. Section 5 presents the performance evaluation results. Finally, Section 6 concludes the paper.

II. Related Works

Most previous studies on flash memory focused on address mapping schemes. Block-level mapping [7] 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 logical block and physical block. Block-level mapping requires only a small-sized mapping table. However, even when only a small portion of a block should be updated, all the non-updated pages should be copied to other clean block invoking a significant page copying overhead.

In page-level mapping [8], a logical page address is mapped to a physical page address. Due to the independent management of pages, page-level mapping is more efficient than the block-level mapping, but it requires a large memory space for the mapping table.

Hybrid mapping [9, 10, 11] uses both the page-level mapping and the block-level mapping. It reserves a portion of flash blocks as a log buffer. Hence, the FTLs with hybrid mapping 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 the valid data in a log block is moved into data blocks to make free space. FTLs using hybrid mapping can yield high performance with a small-sized mapping table. Therefore, most of FTLs employ the hybrid mapping technique.

There are several studies on log buffer-based FTLs. In the block-associative sector translation (BAST) [9] scheme, a log block is associated with only one data block, that is, when the pages in a data bock are updated, the new data should be written at the associated log block. The GC is invoked when there are no clean pages in the associated log block or 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 this 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 [10], where one log block can be associated with multiple data blocks. Therefore, it can prevent the frequent GCs for random write requests. However, the FAST scheme requires a large GC cost once garbage collection is invoked because it should move 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. The buffer cache management is important for achieving high performance flash storage since the I/O requests on flash memory change depending on the buffer cache management techniques. Jo *et al.* [12] proposed a flash-aware buffer management scheme, called FAB, that uses block-level buffer replacement policy that evicts all the pages of a logical block at a time to reduce the GC cost. The logical block that has the largest number of pages in the buffer cache is selected as a victim to be replaced in the FAB scheme. Park *et al.* [13] proposed a clean-first LRU (CFLRU) replacement policy that delays the flushing of dirty page in the buffer cache to reduce the number of write requests to the flash memory. Kim *et al.* [14] proposed a BPLRU buffer management scheme that also evicts all the pages of a victim block like FAB, but it determines the victim block based on the block-level LRU value. In addition, BPLRU writes a whole block into a log block using the block padding technique. Therefore, all log blocks can be switched into a data block without page migrations.

Recently, Lee *et al.* [15] have proposed a buffer-aware garbage collection (BA-GC) technique. BA-GC exploits the duplicated pages that are written in both the buffer cache and flash memory. During garbage collection, the duplicated dirty pages are evicted into the flash memory to eliminate unnecessary page migrations.

Li *et al.* [6] proposed a 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 at 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 by 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 can increase the garbage collection overhead of the hybrid mapping FTLs. This is because DA-GC generates more write requests on the log buffer, and thus invokes frequent GCs in the hybrid mapping.

Our proposed techniques are based on the DA-GC scheme. However, the proposed LDA-VBS and LDA-BM techniques solve the problem of DA-GC in the hybrid mapping FTLs by considering the locality of duplicated data.

Jung *et al*. [16] proposed a flash-aware swap system (FASS), where the operating system directly manages the flash swap space to reduce the garbage collection overhead. Kwon *et al*. [17] modified the greedy garbage collection policy [18] for flash swap space to minimize the garbage collection overhead and to prolong the life time of flash memory. Their swap-aware garbage collection assumed that a recently swapped-out page has a high probability to be swapped-in. These flash-aware swapping techniques can be used in combination with our approach.

III. Motivations

In this section, we describe the DA-GC technique and its disadvantage in hybrid mapping. In hybrid mapping, the log blocks are used as the write buffer for data blocks. If an update request for valid pages of data block occurs, then the new data is written in a log block and the old page in the associated data block is invalidated. If there is no free space in the log block, then the GC is invoked.

Figure 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 by the block-level mapping, all pages are written at the specified page offsets within the data block. PBN 5 and PBN 6 are free blocks reserved for garbage collection. We assume that each flash block is composed of four pages. The log blocks have the logical

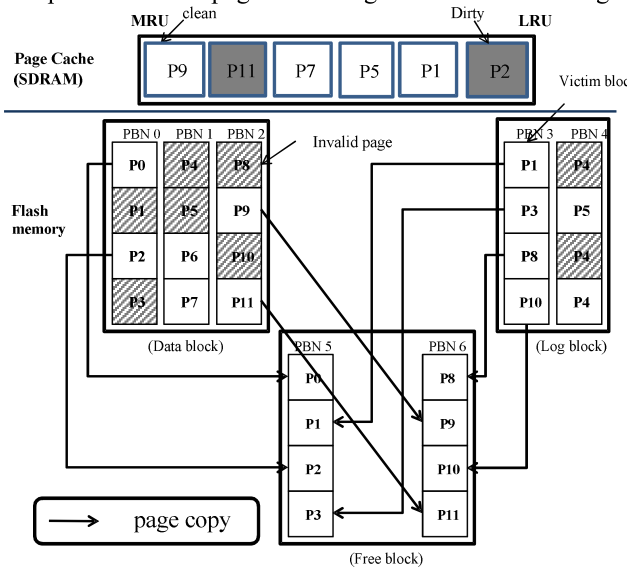

Figure 1. An example of duplication-unaware garbage collection

pages P1, P3, P4, P5, P8, and P10, and the corresponding pages in the data blocks are invalidated. PBN 3 in the log buffer is associated with data blocks PBN 0 and PBN 2. PBN 4 is associated with only PBN 1.

Garbage collection should be invoked since 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, PBN 0, and PBN 2 to the free blocks PBN 5 and PBN 6. The valid pages P0~P3 and P8~P11 are copied to PBN 5 and PBN 6, respectively. Since all valid pages in both the log block and its associated data blocks are merged into free blocks, this step is called the *block merge*. After the block merge is completed, PBN 5 and PBN 6 are changed into data blocks. PBN 0, PBN 2 and PBN 3 are erased, and one of them is allocated for a log block.

The DU-GC does not consider the duplicated pages in the page cache. If we have the information on the page cache, then a more efficient GC can be implemented by considering the duplicated pages that both the page cache and flash memory have. Figure 2 shows the duplication-*aware* garbage collection (DA-GC) scheme [6]. When GC selects PBN 3 as a victim block, GC does not copy the duplicated pages P1, P2, P9, and P11 since the page cache also has these pages. 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.

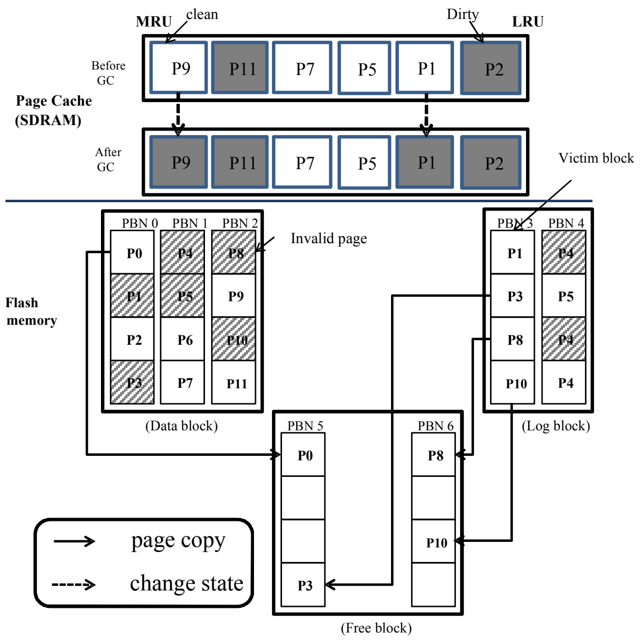

Figure 2. An example of DA-GC

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 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 under page-level mapping. However, DA-GC may invoke frequent garbage collections in hybrid mapping. For instance, in Figure 2, four physical pages in PBN 5 and PBN 6 are not used in DA-GC. Instead, when pages P1 and P2 are evicted from the page cache by page replacement, they should be written in the log blocks. As a result, the log blocks consume the free space more quickly.

If these pages remain clean until they are evicted from the page cache, i.e., GC or host requests does not change these pages into dirty, then these pages will not be written in the flash memory. Especially, since page P1 has not recently been used, there is little possibility for the page to be updated (thus changed to dirty) before it is evicted from the page cache. Therefore, it is better to copy page P1 during the garbage collection. However, since page P9 is the MRU page, it has a high possibility to be changed into dirty even though GC does not change its state. Therefore, excluding page P9 from page migration will not increase the frequency of GC. Consequently, the DA-GC technique 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 region and MRU region, and use different policies for the regions. The LDA-VBS technique avoids selecting the victim log block that includes duplicated clean pages in the LRU region of page cache, as much as possible. The LDA-BM technique copies the duplicated pages during page migration if the corresponding pages in the page cache are clean and are located in the LRU region in order to not change them into dirty. In addition, we propose the LRU dirty page eviction (LDE) technique that enforces the dirty pages in the LRU region to be evicted during garbage collection to reduce the unnecessary page migrations. The proposed techniques can prevent the frequent garbage collections of DA-GC in hybrid mapping while exploiting the advantage of DA-GC that reduces unnecessary copies during garbage collection.

IV. LOCALITY-AWARE GARBAGE COLLECTION

A. *Locality and Duplication-Aware Victim Block Selection*

General victim block selection algorithms consider the block merge cost when selecting a victim block. However, in order to prevent the duplicated clean pages in the LRU region of page cache from being changed into dirty, we should consider not only the merge cost, but also the potential loss resulting from the increase of write requests in 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:

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

where $A(L_i)$ and $\delta(L_i)$ denote the number of data blocks associated with L_i and the number of non-duplicated valid pages in L_i and its associated data blocks (thus they exist only in the flash memory), respectively. For example, in Figure 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 is 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 bocks and one log block are erased. Therefore, $A(L_i)$+1 number of block erases is required.

However, as explained at Section III, if DA-GC is used, then the duplicated clean pages are changed into dirty pages, invoking more write requests from the page cache to the flash memory. Therefore, DA-GC has the potential loss as follows:

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

where $\gamma(L_i)$ represents the number of duplicated pages of the log block L_i whose corresponding pages in page cache are changed from clean into dirty by the DA-GC and are not updated further by following host requests until they are evicted. In Figure 2, two dirty pages are generated by DA-GC. However, since page P9 has a high possibility to be changed to dirty by host requests, the value of $\gamma(L_i)$ will be smaller than 2. The cost for writing the dirty pages in the flash memory is $\gamma(L_i)\cdot C_w$. In addition, the write requests invoke more garbage collections. We add the cost of α that represents the average block merge cost per a dirty page write in order to consider the overhead. The approximated 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 the knowledge of future host requests. To predict these values, we used the 3-region LRU buffer [15], with which we can predict the update probability of each page in the page cache based on the locality information.

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

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

The LDA-VBS technique selects the victim block with the lowest value of $C_{total}(L_i)$. LDA-VBS can then 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 to be 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 page cache have high probabilities to be 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 page cache have low possibilities to be changed to dirty by host requests. Therefore, it may be beneficial to not apply the DA-GC technique to the clean pages in the LRU region. Then, we can prevent the clean data from being changed into dirty data. Though the page migration cost increases during GC, we can reduce the frequency of GC that invokes a large cost.

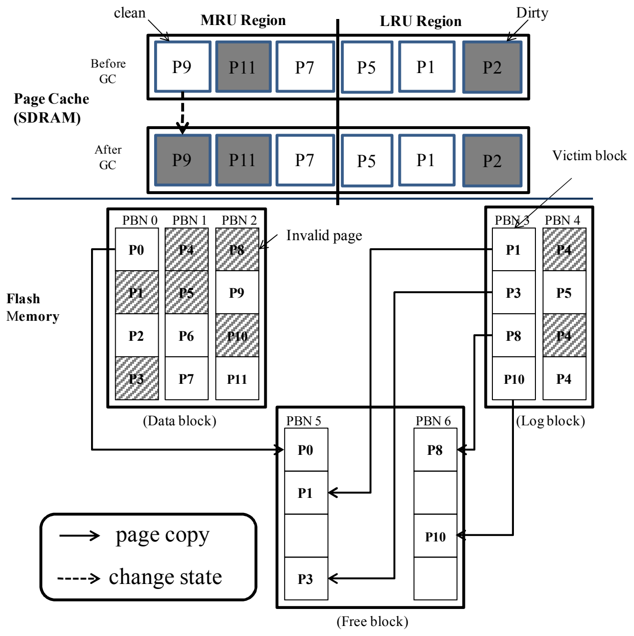

Figure 3. An example of LDA-BM

Figure 3 shows the LDA-BM technique. Page P1 is copied during the block merge operation in the flash memory since the corresponding page in page cache is clean and is located in the LRU region, and P1 in the page cache remains clean. However, the DA-BM technique is applied to the clean page P9 in the MRU region of page cache and to the dirty pages P2 and P11. Even though page P9 is changed to dirty, the potential loss by the page will be small since it has a high possibility of being changed to dirty by host requests.

Under the LDA-BM technique, the victim block selection technique 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 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 copying more pages in the flash memory. 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 when DA-GC uses 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 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 $\gamma_{MRU}(L_i)$.

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 from the page cache to the flash memory during garbage collection because the dirty pages in the LRU region 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 pages unused by DA-GC. The copied dirty pages in the page cache are changed to clean in the page cache.

We can simultaneously use both LDA-BM and LDE techniques during the block merge operation to apply different policies with DA-BM 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 different policies for the pages in the LRU region.

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:

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

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

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

where $\gamma_{LRU}(L_i)$ represents the number of duplicated pages of the log block L_i whose corresponding pages in 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 summarizes the changes of duplicated pages under each scheme. $\rho(L_i)$ denotes the number of duplicated pages of the log block L_i whose corresponding pages in page cache are duplicated in the MRU region. DA-GC and LDA-VBS have the lowest GC costs but have the largest potential loss on future GC costs since they change all the duplicated clean pages in the page cache into dirty pages. LDA-BM has a higher GC cost than DA-GC but has a lower potential loss because it does not change the states of the duplicated clean pages in the LRU region. By using LDE in addition to LDA-BM, the GC cost increases but there is potential benefit since the duplicated dirty pages in the LRU region are changed into clean pages.

TABLE I.
THE CHANGES OF BUFFER STATE IN EACH SCHEME

Cache Area	Before GC	Numbers	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

$N_{page} = \rho(L_i) + \theta(L_i) + \pi(L_i) + \delta(L_i)$

V. EXPERIMENTAL RESULT

A. *Experiment Environments*

We have implemented a trace-driven simulator in order to evaluate the performances of the proposed schemes. Our simulator consists of the page cache and the flash memory. The flash memory model used in the simulation is based on Samsung SLC large block NAND flash memory [19]. Each flash block is composed of 64, 2 KB pages. The access times 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 page cache is implemented using the 3-region LRU buffer algorithm [15] to divide it into the LRU and MRU regions.

We used five real virtual memory traces collected by the Valgrind 3.4.1 toolset [20]. They are captured while executing the applications in Table II on a Linux system.

The seven schemes, shown in Table III, are compared. Each scheme uses different victim block selection and block merge techniques. All schemes use the FAST hybrid mapping FTL. We assumed that the normal VBS algorithm is the round-robin selection policy, which selects the oldest log block as a victim. Since the oldest blocks generally have a small number of valid pages, the round-robin selection policy invokes small GC overheads.

TABLE II.
THE CHARACTERISTICS OF THE WORKLOAD USED IN EXPERIMENTS

applications	write ratio	scenario
acrobat	23.1%	view PDF file
gqview	21.1%	picture modification after viewing image file
kword	14.9%	data modification
mozilla	13.2%	web information search (google, yahoo, amazon etc.)
office	19.7%	slide show

TABLE III.
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 Analysis*

Figure 4 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 and write 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 is similar or inferior to that of DU-GC because the potential loss of DA-GC is larger than the GC cost reduction resulting from not copying the duplicated pages. From the results of LDA-GC$_1$ and LDA-GC$_2$, we can know that the LDA-VBS technique

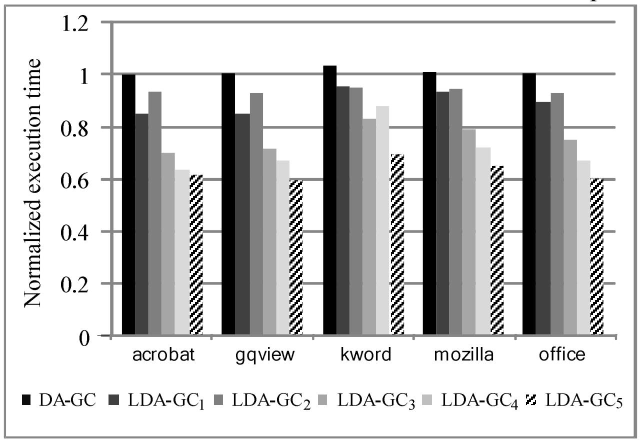

Figure 4. The total execution time comparison

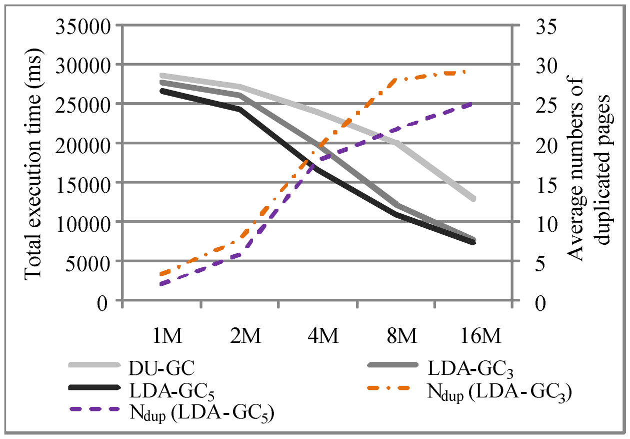

Figure 5. The execution times and average numbers of duplicated pages while varying the page cache size (kword workload)

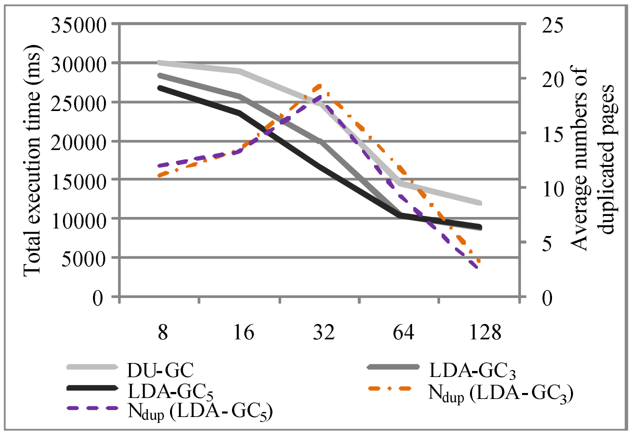

Figure 6. The execution times and average numbers of duplicated pages while varying the number of log blocks (kword workload)

(which presents the performance improvement by 10% on average) is more effective than the LDA-BM technique (which presents the performance improvement by 6% on average). This is because LDA-VBS can significantly reduce the garbage collection overhead as well as the potential loss.

The LDA-GC$_3$ scheme, which uses both LDA-VBS and LDA-BM, shows 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, improves the performance by 28% on average. The LDA-GC$_5$ scheme, which uses all the proposed techniques, reduces the I/O execution times by 37% on average compared to that of DU-GC.

We also evaluated the effect of page cache size. Figure 5 illustrates the total I/O execution time and the average number of duplicated pages (N_{dup}) for kword workload while varying the size of the page cache from 1 MB to 16 MB. The number of log blocks is fixed to 32. The performances are improved as the size of page cache increases because the hit ratio of page cache increases. LDA-GC$_3$ and LDA-GC$_5$ show better performances than does DU-GC regardless of the page cache size. Moreover, as the size of the page cache increases, the performance gaps between DU-GC and LDA-GC schemes are increased since the number of duplicated pages increases as shown in Figure 5. When there are many duplicated pages, the proposed schemes have more chances to reduce the garbage collection overhead.

We also evaluated the effect of the flash log buffer size. Figure 6 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 is fixed to 4 MB. As the number of log blocks increases, the execution times are reduced. When there are many log blocks, it takes a long time 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 schemes increase as the number of log blocks increases. This is because LDA-VBS can find a more proper victim block when there are many log blocks available.

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

Figure 7 shows the number of garbage collections invoked during benchmark executions under the proposed GC schemes. These values are normalized by those of DU-GC. While LDA-GC$_4$ and LDA-GC$_5$ that use the LDE technique outperform DU-GC by about 4~5%, other schemes, i.e., DA-GC, LDA-GC$_1$, LDA-GC$_2$, and LDA-GC$_3$, invoke larger numbers of GCs than does DU-GC due to the potential loss on GC cost. Since all proposed schemes show the performance improvements over DU-GC as shown in Figure 4, we can infer that each GC invocation requires smaller cost under the proposed GC schemes than DU-GC.

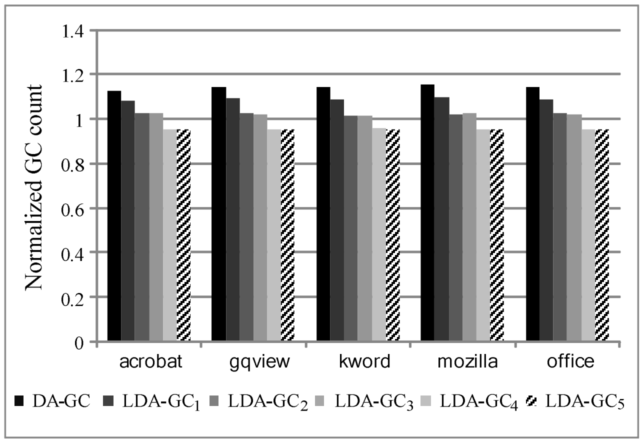

Figure 7. The number of garbage collections

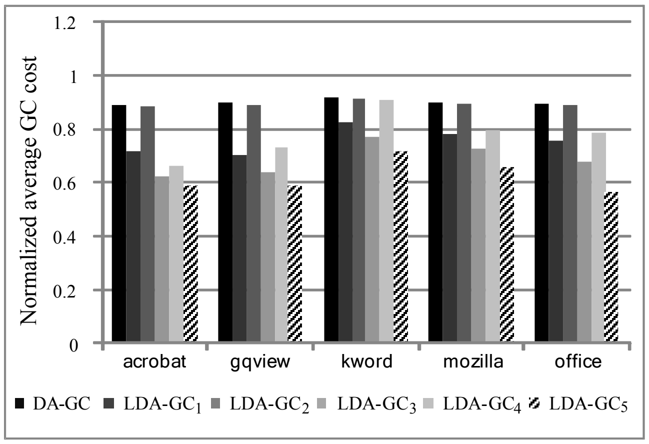

Figure 8. The average garbage collection cost

Accordingly, we measured the average cost per garbage collection under the proposed GC schemes as shown in Figure 8. These values are normalized by those of DU-GC. All GC schemes provide smaller average GC costs than does DU-GC. Especially, the average GC costs of LDA-GC$_1$, LDA-GC$_3$, and LDA-GC$_5$, which use the LDA-VBS technique, are lower than those of other schemes since LDA-VBS considers the GC cost of the victim log block. Although LDA-GC$_4$ and LDA-GC$_5$ have similar numbers of garbage collections in Figure 7, LDA-GC$_5$ achieves better performance than LDA-GC$_4$ due to the lower average GC cost as shown in Figure 8.

VI. CONCLUSION

Flash memory is a good device for swap space of virtual memory systems. For the flash memory-based virtual memory systems, the locality and duplication-aware garbage collection technique are proposed, which can reduce the 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 the hybrid mapping FTLs, the proposed locality and duplicated-aware victim block selection (LDA-VBS) technique considers both the garbage collection overhead and the potential loss. The locality and duplicated-aware block merge (LDA-BM) and LRU dirty page eviction (LDE) techniques selectively apply the duplication-aware page migration depending on the locality of each page in the page cache. The experimental results showed that the LDA-VBS and LDA-BM techniques reduce the total I/O execution time by 10% and 6%, on average, compared to that of DU-GC, respectively. By applying both the LDA-VBS and LDA-BM techniques, the performance can be improved by 24% on average compared to that of DU-GC. By additionally adopting the LDE technique, performance can be improved by 37%, on average, compared to that of DU-GC.

REFERENCES

[1] C. Park, J. Kang, S. Park, and J. Kim, "Energy-Aware Demand Paging on NAND Flash-based Embedded Storages," In Proc. International Symposium on Low Power Electronics and Design, pages 338–343, 2004.

[2] Y. Joo, Y. Choi, C. Park, S. W. Chung, E.-Y. Chung, and N. Chang, "Demand Paging for OneNAND Flash eXecute-In-Place," In Proc. CODES+ISSS'06, pages 229–234, 2006.

[3] J. In, I. Shin, and H. Kim, "SWL: A Search-While-Load Demand Paging Scheme with NAND Flash Memory," In Proc. Conference on Languages, Compilers, and Tools for Embedded Systemsn (LCTES), pages 217–225, 2007.

[4] Intel Corporation, "Understanding the flash translation layer (FTL) specification," http://developer.intel.com/.

[5] CompactFlash Association, http://www.compactflash.org/.

[6] H.-L. Li, C-L. Yang, H-W. Tseng, "Energy-Aware Flash Memory Management in Virtual Memory System," IEEE Trans. Very Large Scale Integration System, 16(8):952–964, 2008.

[7] A. Ban. Flash file system optimized for page-mode flash technologies, US Patent 5,937,425, Aug. 10, 1999.

[8] A. Ban. Flash file system, US Patent 5,404,485, Apr. 4, 1995.

[9] 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, vol. 48, no. 2, pp. 366-375, 2002.

[10] S. W. Lee, D. J. Park, T. S. Chung, W. K. Choi, D. H. Lee, S. W. Park, and H. J. Song, "A Log Buffer based Flash Translation Layer using Fully Associative Sector Translation," ACM Trans. on Embedded Computing Systems, vol. 6, no. 3, 2007.

[11] C. I. Park, W. M. Cheon, J. G. Kang, K. H. Roh, W. H. Cho, and J. S. Kim, "A Reconfigurable FTL (Flash Translation Layer) Architecture for NAND Flash-based Applications," ACM Trans. on Embedded Computing Systems, vol. 7, no. 4, 2008.

[12] 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, vol. 52, no. 2, pp. 485-493, 2006.

[13] S. Y. Park, D. Jung, J. U. Kang, J. S. Kim, and J. Lee, "CFLRU: A Replacement Algorithm for Flash Memory," In Proc. of the Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems, pp. 234-241, 2006.

[14] H. Kim and S. Ahn, "BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage," In Proc. of the USENIX Conf. on File and Storage Technologies, pp. 239- 252, 2008.

[15] S. Lee, D. Shin, and J. Kim, "Buffer-Aware Garbage Collection Techniques for NAND Flash Memory-Based Storage Systems," In Proc. of the Int. Work. on Software Support for Portable Storage (IWSSPS'08) , pp.27-32, 2008.

[16] D. W. Jung, J. S. Kim, S. Y. Park, J. U. Kang, and J. Lee, "FASS: A Flash-Aware Swap System," In Proc. of the Int. Work. on Software Support for Portable Storage (IWSSPS'05), 2005.

[17] O. Kwon, K. Koh, "Swap-Aware Garbage Collection for NAND Flash Memory Based Embedded Systems," In Proc. of the Seventh IEEE Int. Conf. on Computer and Information Technology, 2007.

[18] M. Wu and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," In Proc. of the Int. Conf. on Architectural Support for Programming Languages and Operating Systems, pp.86-97, 1994.

[19] Samsung Corp, "K9WBG08U1M NAND Flash Memory," 2007.

[20] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *SIGPLAN Not.*, vol. 42, no. 6, pp.89–100, 2007.