Recently-Evicted-First Buffer Replacement Policy for Flash Storage Devices

Dongyoung Seo and Dongkun Shin, *Member*, IEEE

**Abstract — ***Flash memory has been widely used as a storage device for consumer devices. Recently, applications using flash memory are becoming complex and diverse. One of obstacles to its wide use is the slow write performance of flash memory caused by its erase-before-write characteristic. To enhance the write performance, FTL (Flash Translation Layer) generally uses the flash log-buffer, where data is written by the out-of-place scheme. However, current log buffer-based FTL schemes show poor performance for random write requests due to the block thrashing problem and high block associativity. Recently, flash-aware buffer cache management schemes are proposed to solve the block thrashing problem. However, they cannot also eliminate the problem completely. In this paper, we propose a novel flash-aware buffer cache replacement policy. The technique selects the victim page to be evicted from buffer cache considering the recent victim page sent to the flash log buffer. Our experimental results show that the proposed technique reduces the flash I/O execution time by up to 30%*[1].

Index Terms — flash memory, buffer management, page replacement, flash translation layer, embedded storage.

I. INTRODUCTION

Since NAND Flash memory has many advantages over hard disk drive such as low-power consumption, small size and high shock resistance, it has been widely used for mobile consumer devices such as MP3 player, digital camera, personal digital assistant and cell phone. In the past, the usage scenarios of flash memory were simple and regular. For example, MP3 player and digital camera used the NAND flash memory to read and write only large-sized multimedia files.

However, recent applications for flash memory are complex and diverse. For instance, current convergence of consumer devices enables recent cell phones to offer e-mail service and full Web browsing. To do that, they should store many temporary internet files or small-sized e-mail files on the flash memory. Moreover, these applications can be executed concurrently thus generating mixed write requests. Recently, general purpose systems such as desktop PC are also going to use flash memory. For example, hybrid hard disk [1] and on-board disk cache [2] use the flash memory as a nonvolatile cache of hard disk drive. NAND flash-based solid-state disk (SSD) is expected to replace hard disk in the near future [3, 4]. Such changes on flash memory applications require a more efficient management scheme for flash memory.

Flash memory provides three operations: read, write and erase. The read performance of flash memory is high because it requires no seek time. However, flash memory has a low write performance due to its "erase-before-write" constraint which means that a block should be erased before a data is written into the block. While the write operation is performed by the unit of page, the erase operation should be performed by the unit of block which is composed of several pages. For example, in the large block NAND flash memory, a page is 2 KB and a block is 128 KB (64 pages).

To handle the special features of flash memory, most systems use flash translation layer (FTL) which maps the logical page address from the file system to the physical page address in flash memory devices. The address mapping schemes of FTL can be divided into three classes, i.e., block-level mapping, page-level mapping and hybrid mapping.

In the block-level mapping, the mapping table maintains the mapping information between logical block address and physical block address. So, a logical page should be written by the *in-place* scheme, which means a page is written at the fixed location of a block determined by the page offset within a block. The block-level mapping needs a small-sized mapping table. However, even when only a small portion of a block should be modified, the specified block should be erased and the nonupdated pages as well as the updated page should be copied into a new block. This constraint results in a high page migration cost thus poor write performance.

In the page-level mapping, the mapping table maintains the mapping information between logical page address and physical page address. Therefore, a logical page can be mapped by the *out-of-place* scheme, which means a logical page can be written to any physical page in a block. If an update request is sent for a data which is already written in flash memory, FTL writes the new data to a different clean page and changes the page-level mapping information. The old page is invalidated by marking in the spare field of the page. The drawback of page-level mapping is that the mapping table size is inevitably large.

The hybrid mapping uses both page mapping and block mapping. In this scheme, all the physical flash memory blocks are separated into log blocks and data blocks. The log blocks are called log buffer. So, the FTL using hybrid mapping scheme is called as a *log buffer-based FTL*. While the log

[1] This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD). (KRF-2007-331-D00358)

D. Seo is with the Samsung Electronics, Suwon, Korea (e-mail: dongyoung.seo@gmail.com).

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

Contributed Paper
Manuscript received July 9, 2008 0098 3063/08/$20.00 © 2008 IEEE

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.

blocks use the page-level mapping and the out-of-place scheme, the data blocks are handled by the block-level mapping and the in-place scheme.

For a write request, the hybrid mapping FTL first sends the data to a log block invalidating the corresponding old data in data block. If the log blocks are full and there is no empty space, one log block is selected as a victim and all the valid pages in the log block are migrated into data blocks to make a room for on-going write requests. In this step, the log block is merged with the data blocks which are associated with it. So, this step is called block merge. There are three kinds of block merges: *full merge, partial merge* and *switch merge* [5]. The partial merge and switch merge can be done only when all the pages in the victim log block are written by the in-place scheme. While the full merge requires many page copies and block erases, the partial merge and switch merge invoke low page migration costs. The hybrid mapping can reduce the page migration cost compared to the block mapping with a small-sized mapping table.

To enhance the write performance of flash memory system, the overhead invoked by the block merge should be reduced. Therefore, most existing log-buffer based FTL schemes aim to reduce the number of block merge. However, current FTL techniques are mainly focusing on the sequential write pattern since they target the multimedia systems such as MP3 player and digital camera. However, in the recent flash memory devices, multiple processes generate both sequential and random write requests concurrently. Therefore, most of current FTL techniques will show poor performance since random writes incur frequent log block merges.

Another technique to improve the write performance of flash memory is to use the buffer cache. The buffer cache is volatile memory located between the file system and the flash memory. The buffer cache can reduce the number of write requests sent to the flash memory by merging repeated write requests on the same page. The buffer replacement policy of buffer cache is an important issue since it determines the flash memory write pattern.

In this paper, we propose a novel buffer replacement policy which considers the log buffer in flash memory. The policy is called REF (Recently-Evicted-First) since it gives high priorities to the pages of the block whose pages are recently evicted to the flash log buffer. It is similar to the I/O scheduler for hard disk drive in Linux system. The I/O scheduler reorders and clusters the write requests to reduce the average number of head movement in hard disk [6]. The REF also clusters the write requests in buffer cache to reduce the number of block merge. The REF policy improves the performance of flash memory by 20%-30% compared with the LRU (Least-Recently-Used) buffer replacement policy for benchmarks.

The rest of the paper is organized as follows. In Section 2, the related works and their drawbacks are introduced. Section 3 describes the details of REF buffer cache management scheme. Section 4 introduces the BP-REF scheme which is an extended version of REF. Experimental results are presented in Section 5. Section 6 concludes with a summary and future works.

II. RELATED WORKS

A. Log Buffer-Based FTL

There have been many researches on the log buffer-based FTLs. There are two kinds of schemes depending on the block association policy as shown in Fig. 1, i.e., 1:1 log block mapping (BAST) [5] and 1:N log block mapping (FAST) [7]. The block association policy means how many data blocks a log block can be used for. In the 1:1 scheme, a log block is allocated for only one data block. In Fig. 1(a), there are 5 data blocks (B0, B1, B2, B3 and B4) and 2 log blocks (L0 and L1). We assume that each block consists of four pages. When the update requests on the pages p0 and p4 come, the pages are written at the log blocks invalidating the pages in the data blocks. (The grey-colored pages are invalid pages.) The log blocks L0 and L1 are associated with the data blocks B0 and B1 respectively as shown in Fig. 1(a) where the arrow lines indicate the block association. The pages p0 and p4 are written into their associated log blocks respectively.

The 1:1 log block mapping of BAST can invoke frequent log block merges. For instance, in Fig. 1, if the write sequence "p8, p12, p1, p5, p9, p13" comes, BAST should replace either of the two log blocks generating an expensive merge operation for every write because there is no log block allocated for the target data block. Moreover, every victim log block in this example holds only one page when it is replaced; the other three pages remain empty. So, the log blocks in BAST would show very low space utilization when they are replaced from the log buffer. If the write request pattern is random, the 1:1 mapping scheme shows poor performance since frequent log block merges are inevitable. Such a phenomenon where most write requests invoke a block merge is called *log block thrashing*.

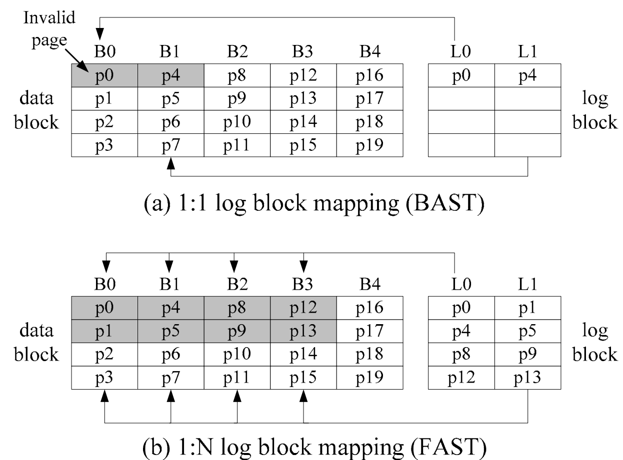


Fig. 1. Log buffer-based FTL schemes.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.

To prevent the log block thrashing problem, the 1:N mapping scheme of FAST was proposed. In 1:N scheme, a log block can be used for multiple data blocks at a time as shown in Fig. 1(b). Using the 1:N mapping, we can prevent the log block thrashing problem. For the same write sequence "p0, p4, p8, p12, p1, p5, p9, p13", there is no block merge in FAST whereas each page write incurs a block merge in BAST. However, the problem of 1:N mapping is its high block associativity, where the block associativity means how many data blocks are associated with a log block. For example, when the log block L1 in Fig. 1(b) is replaced from the log buffer, 16 page copies (16 pages = 4 blocks × 4 pages) are required since the log block L1 is associated with four data blocks, B0, B1, B2 and B3. This means that FAST scheme requires a large cost per block merge though it invokes a small number of block merge. The maximum block associativity is same to the number of pages in a block.

Recently, a special N:N scheme was introduced, where N number of log blocks can be used for N number of data blocks. Superblock scheme [8] is one example of N:N mapping. The N:N scheme is a hybrid form of 1:1 mapping scheme and 1:N mapping scheme. So, it also has both the block thrashing problem and the high block associativity problem.

B. Flash-Aware Buffer Schemes

There are several researches on buffer cache management scheme aiming to reduce the flash memory write cost.

Park et al. [9] proposed a clean-first LRU (CFLRU) replacement policy which delays the flush of dirty page in the buffer cache to reduce the number of write request to the flash memory. Jo et al. [10] proposed a flash-aware buffer management scheme, called FAB. Using block-level buffer replacement which evicts all the pages of a block at a time, it reduces the block merge cost. FAB scheme first finds a block which has the largest number of pages in the buffer cache. Then, all the pages of the block are flushed into the flash memory. Kim et al. [11] proposed a BPLRU (Block Padding Least Recently Used) buffer management scheme. It 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 by the in-place scheme using the block padding technique. Therefore, all log blocks can be merged by the switch merge which requires no page migration.

The log block thrashing and high block associativity problems of log buffer-based FTL can be diminished using the flash-aware buffer techniques such as FAB and BPLRU. Since such techniques flush all the pages of a block into the log block at a time, the subsequent page writes do not invoke the log block replacement and do not increase the block associativity of a log block.

However, these schemes cannot also avoid the block thrashing problem completely. For example, in Fig. 2, there are 9 pages in the buffer cache. FAB scheme manages the block node list which is a linked list of blocks sorted by their recency that means how recently a block is accessed. Each block node points the page node list which is a list of pages that belong to the corresponding block. When the buffer is full and cannot accommodate a new page data, the buffer should evict a number of pages. The block which has the largest number of pages is selected in FAB. Therefore, the block B4 is selected. (If more than one block has the same number of pages, the block which is not accessed for the longest time is selected.) Since two log blocks L0 and L1 have already been assigned to the data blocks B0 and B1, the log block merge should be performed in the BAST scheme. The subsequent block flushes also invoke the block merge as shown in Fig. 2.

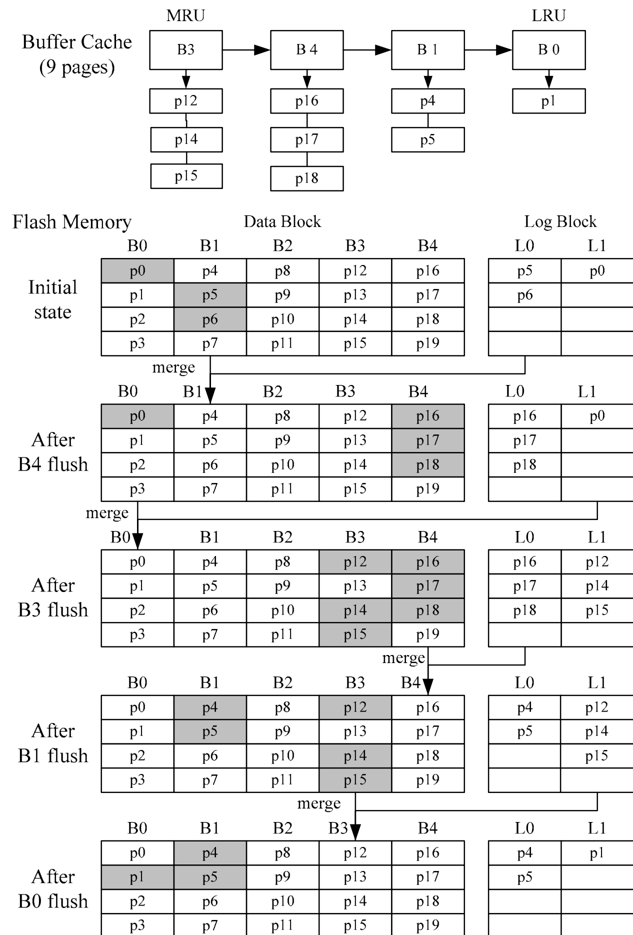


Fig. 2. Block thrashing under FAB scheme.

Another problem of FAB is that it can evict the recently used pages if the corresponding block has the largest number of pages in the buffer cache. For example, in Fig. 2, even though the blocks B3 and B4 are recently used, the blocks are evicted prior to the blocks B1 and B0. If the pages of B3 and B4 are updated just after the eviction, the page eviction turns out to be useless. This problem results from that the block-level page eviction is prior to the page recency in selecting a victim page in FAB.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.

BPLRU also manages all the pages in buffer cache by the block level LRU policy. The block which is not accessed for the longest time is selected as a victim block. BPLRU also has the block thrashing problem. One difference is that BPLRU invokes only switch merges thanks to the block padding. However, BPLRU has an opposite problem in handling the page recency. If one of pages in a block has a high recency (i.e., recently used), other pages in the block also stay in the buffer cache even though the pages are not recently used. Then, the not-recently-used pages waste the space of buffer cache. The block-level management of page recency is a critical weak point of FAB and BPLRU.

In summary, current FTL schemes and current flash-aware buffer management schemes can show poor performance for random writes due to the block thrashing problem.

III. LOG BUFFER-AWARE BUFFER MANAGEMENT

A. REF Page Eviction

The previous buffer management schemes have no consideration on the log buffer in the flash memory. If the buffer cache selects a victim page to be evicted such that the corresponding data block of the victim page is associated with the current log blocks, we can avoid the log block thrashing and the high block associativity problems.

The proposed REF scheme has three main features as follows:

- Block-level page eviction: Select victim blocks and evict only the pages of the victim blocks from buffer cache to reduce the number of block merges or block associativity. This is improvement over the LRU replacement policy.
- Log buffer-aware victim selection: As far as possible, the victim blocks are maintained such that they are same to the data blocks associated with the log blocks. This is improvement over FAB and BPLRU policies.
- Page-level recency consideration: The victim pages are selected among the not-recently-used pages to prevent the recently-used pages from being evicted. This is improvement over FAB and BPLRU policies.

To select the victim blocks which are associated with the log blocks, the REF scheme determines them considering the recent page eviction. For instance, assume that a buffer cache has the pages p0, p4, p9, p13, p1 and p5 in the LRU order, two log buffers has the pages p8 and p12 and the log buffer is managed by the BAST scheme as shown in Fig. 3. If we use the LRU page replacement policy, the pages are evicted by the order of “p0, p4, p9, p13, p1, p5” and 6 number of log block merges are invoked. However, if we reorder the page eviction, we can reduce the number of block merges. Since the log blocks are associated with the data blocks B2 and B3, it is better to evict the pages of B2 and B3 first, i.e., p9 and p13. So, if the buffer cache flushes the pages by the sequence of “p9, p13, p0, p4, p1, p5”, only two number of log block merges are required in the BAST scheme.

For the FAST scheme, the block associativity of each log block can be reduced by REF. Fig. 4 shows the changes of log blocks after the pages in the buffer cache of Fig. 3 are flushed into the flash memory managed by the FAST scheme. In the initial log blocks, both the pages p8 and p12 are written at the log block L0 when the FAST scheme is used. If the LRU page replacement policy is used, both the log blocks L0 and L1 have the block associativity of 4. However, if REF policy is used, the block associativity is reduced to 2 for both log blocks. Therefore, we can know that REF scheme can reduce the block associativity of log block as well as the number of block merges.

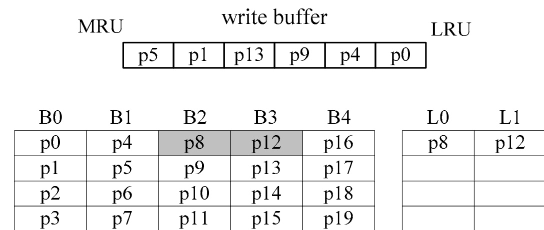


Fig. 3. LRU eviction vs. REF eviction in BAST scheme.

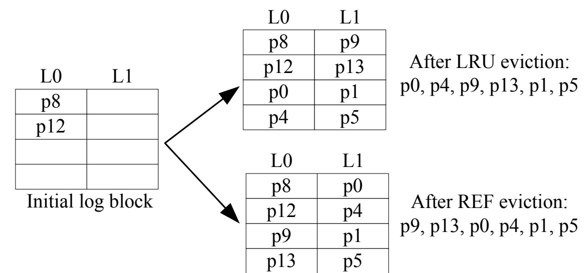


Fig. 4. LRU eviction vs. REF eviction in FAST scheme.

B. Victim Block Selection

To evict only the pages whose corresponding data blocks are associated with the log blocks, the buffer manager should know the status of log blocks exactly. To do that, the FTL should provide an interface inquiring the log block status. However, the proposed REF scheme can be implemented without changing the current FTLs. Instead of directly referring to the status of log blocks, the buffer manager selects the victim page using the recent history of buffer eviction.

Fig. 5 shows the page eviction in REF. Assume that the buffer cache can contain 8 pages and each page is sorted in the order of LRU. REF maintains the set of victim block (VB). REF enforces the buffer cache to evict only the pages of victim blocks. The number of VB should be smaller than the number of log blocks in flash memory to prevent the log block thrashing. In this example, the size of VB is 2 (|VB| = 2). REF selects the blocks to be included into VB using the victim window (VW) to prevent the recently-used pages from being evicted. In this example, the size of VW is 75%. So, the six

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.

(75% of 8 pages) number of least recently-used pages are candidates for the victim page. REF finds two blocks which has the largest number of pages within the victim window. First, the blocks B2 and B3 are selected as the victim blocks. Then, REF composes the victim page list with all the pages which are located within the VW and whose corresponding block is in the VB.

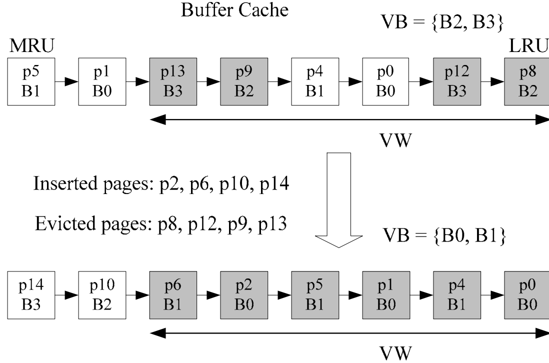


Fig. 5. Victim block selection in REF.

If a free space is required in the buffer cache, the least-recently-used page in the victim page list is evicted. If there is a page whose corresponding block is in VB but which is not within the victim window, the page may enter the victim window after the eviction of other pages. Then, the victim page list is updated due to the insertion of a new victim page.

In Fig. 5, when the new pages p2, p6, p10 and p14 are inserted into the buffer cache, the pages p8, p12, p9 and p13 are evicted in sequence. When there is no page in the victim page list because all the victim pages are flushed into the flash memory, a new VB should be constructed. By the insertion of the pages p2, p6, p10 and p14, the victim blocks are changed to B0 and B1 in Fig. 5.

While FAB and BPLRU evict all the pages of victim block at a time, REF evicts the victim pages only by the amount of the required free space. Since REF evicts only the pages of the predetermined victim blocks, the effect of one-by-one page eviction is similar to that of simultaneous victim pages eviction. Instead, the one-by-one page eviction can reduce the number of buffer cache misses when the update request on the victim pages comes, thus reducing the number of write requests on flash memory.

The size of victim window should be selected carefully considering the locality of write pattern. If the size of VW is too large, recently-used pages are evicted thus increasing the miss ratio of buffer cache. If the size of VW is too small, REF acts like the original LRU scheme thus invoking the log block thrashing. From the experiments using desktop benchmark applications, we observed that the proper victim window size is about 75% of the total size of buffer cache.

Fig. 6 compares the behaviors of buffer cache and log blocks under LRU, FAB, BPLRU and REF policies using an example write sequence. We assumed that the buffer cache can contain 3 pages and the two log blocks of flash memory are managed by the BAST FTL scheme. For each write request sent to the buffer cache, the events on the buffer cache and the log buffer of flash memory are shown. At the column of Cache, there are I and E events which mean the insertion into the buffer cache and the eviction into the flash memory, respectively. At the column of Log Block, there are L and M events which mean the insertion into the log block and the block merge, respectively. The REF scheme generates no block merge while LRU, FAB and BPLRU invoke 2 or 3 number of block merges respectively.

I/O Request	LRU		FAB		BPLRU		REF	
	Cache	Log Block	Cache	Log Block	Cache	Log Block	Cache	Log Block
Write [p0] (B0) Write [p4] (B1) Write [p8] (B2)	I [p0] I [p4] I [p8]		I [p0] I [p4] I [p8]		I [p0] I [p4] I [p8]		I [p0] I [p4] I [p8]	
Write [p5] (B1)	E [p0] I [p5]	L0 [p0]	E [p0] I [p5]	L0 [p0]	E [p0] I [p5]	L0 [p0] L0 [p1] L0 [p2] L0 [p3]	E [p0] I [p5]	L0 [p0]
Write [p9] (B2)	E [p4] I [p9]	L1 [p4]	E [p4] E [p5] I [p9]	L1 [p4] L1 [p5]	E [p8] I [p9]	L1 [p8] L1 [p9] L1 [p10] L1 [p11]	E [p4] I [p9]	L1 [p4]
Write [p1] (B0)	E [p8] I [p1]	**M [L0]** L0 [p8]	I [p1]		E [p4] E [p5] I [p1]	**M [L0]** L0 [p4] L0 [p5] L0 [p6] L0 [p7]	E [p5] I [p1]	L1 [p5]
Write [p10] (B2)	E [p5] I [p10]	L1 [p5]	E [p8] E [p9] I [p10]	**M [L0]** L0 [p8] L0 [p9]	I [p10]		E [p1] I [p10]	L0 [p1]
Write [p2] (B0)	E [p9] I [p2]	L0 [p9]	I [p2]		E [p1] I [p2]	**M [L1]** L1 [p0] L1 [p1] L1 [p2] L1 [p4]	E [p2]	L0 [p2]
Write [p6] (B1)	E [p1] I [p6]	**M [L1]** L1 [p1]	E [p1] E [p2] I [p6]	**M [L1]** L1 [p1] L1 [p2]	E [p9] E [p10] I [p6]	**M [L0]** L0 [p8] L0 [p9] L0 [p10] L0 [p11]	E [p6]	L1 [p6]

I : Insertion E : Eviction M : Block Merge

Fig. 6. Comparison among LRU, FAB, BPLRU and REF. (2 log blocks, 4 pages per a block, 3 page-sized buffer cache, VW= 100%, |VB| = 2)

IV. Block Padding REF

We also propose the block padding REF (BP-REF) which uses the block padding technique in addition to REF. BPLRU shows a good performance using the block padding technique. However, the block padding technique can invoke a large overhead cost since it reads the nonupdated pages from the data block into the buffer cache in order to write a complete block in a log block. For example, in Fig. 6, when the write request on p5 comes, BPLRU writes the pages p1, p2 and p3 (underlined) as well as the victim page p0.

Actually, if the block padding is used, all the pages in a log block are written by the in-place scheme. So, BPLRU makes the hybrid mapping FTL to act like the block-level mapping FTL. Therefore, it inherently has the overhead of block-level mapping. Especially, when only small portion of a block is frequently updated (hot data), most of cold pages in a block should be copied frequently though they are unchanged. When the size of buffer cache is small, this problem is more critical since even hot pages cannot stay for a long time in buffer cache.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.

To reduce the block padding overhead, BP-REF performs the block padding technique *selectively*. Only when the number of victim pages of a victim block is larger than a threshold value, called *block padding threshold*, it performs the block padding to reduce the overhead. For example, when the block padding threshold is 80% and the buffer cache has more than 80% of the total page number of a block, BP-REF reads the rest 20% of the block from the flash memory and evicts the complete block to the log buffer. The proper value of block padding threshold depends on the storage access pattern and the FTL technique as will be shown in Section V.

Fig. 7 shows the example behavior of BP-REF. The selected victim blocks are B1 and B2. When the pages of B1 are evicted, BP-REF uses the block padding. So, it reads the pages p14 and p15 into the buffer cache and evicts all the pages of B1 at a time to the log block L0. This prevents the pages of B2 from being written into the log block L0 thus enabling FTL to do the switch merge.

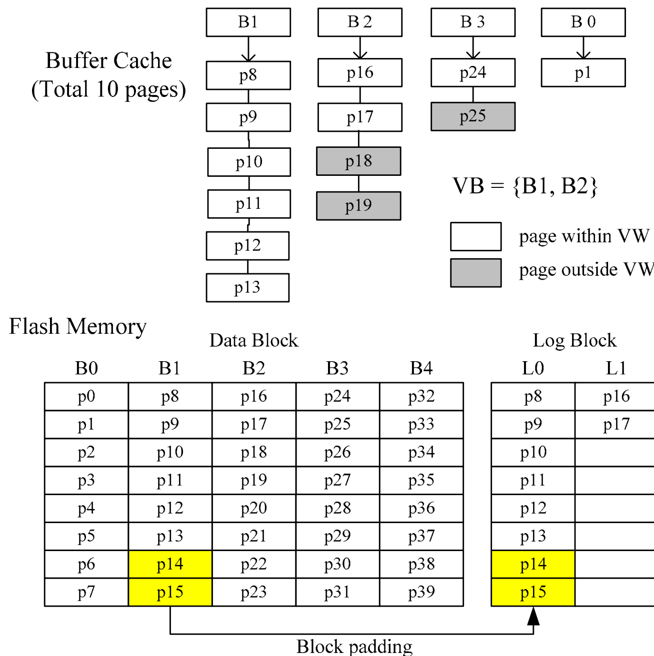


Fig. 7. Block padding in BP-REF.

When the number of victim pages of a victim block is small, it is better not to perform the block padding. Note that REF updates the victim page list when a new page enters the victim window. So, it is better to wait the write requests on other pages of the victim block instead of padding a large number of pages. In Fig. 7, the eviction of B2 does not incur the block padding. If the pages p18 and p19 of B2 are inserted into the victim window after the eviction of B1, we can write the up-to-date pages into the log block L1. Especially under the FAST scheme, BP-REF improves the performance significantly over REF since it reduces the block associativity of log blocks.

V. Experiments

We evaluated the performance of REF using simulation. We collected the disk I/O traces executing several Windows desktop applications under the NTFS file system and used the traces as inputs of the simulation. We used three benchmark programs: Internet Explorer which generates many small-sized temporary files, MS-Office install application which writes both small- and large-sized office files, and JPEG file copy application which copies total 2GB of JPEG files, where each file size is 500~600KB. While Internet Explorer generates highly random write requests, MS-Office install application and JPEG copy application generate both sequential writes and random writes because they update the metadata of NTFS file system. So, there are many random writes which are interposed between sequential writes. The reason why we selected the desktop applications as benchmarks is that current consumer devices begin to offer these applications as a result of convergence. Table 1 shows important parameters used in the simulation. The page size, the block size and the timing values of flash memory are based on the specification of large block NAND flash memory.

TABLE I
Simulation Parameters

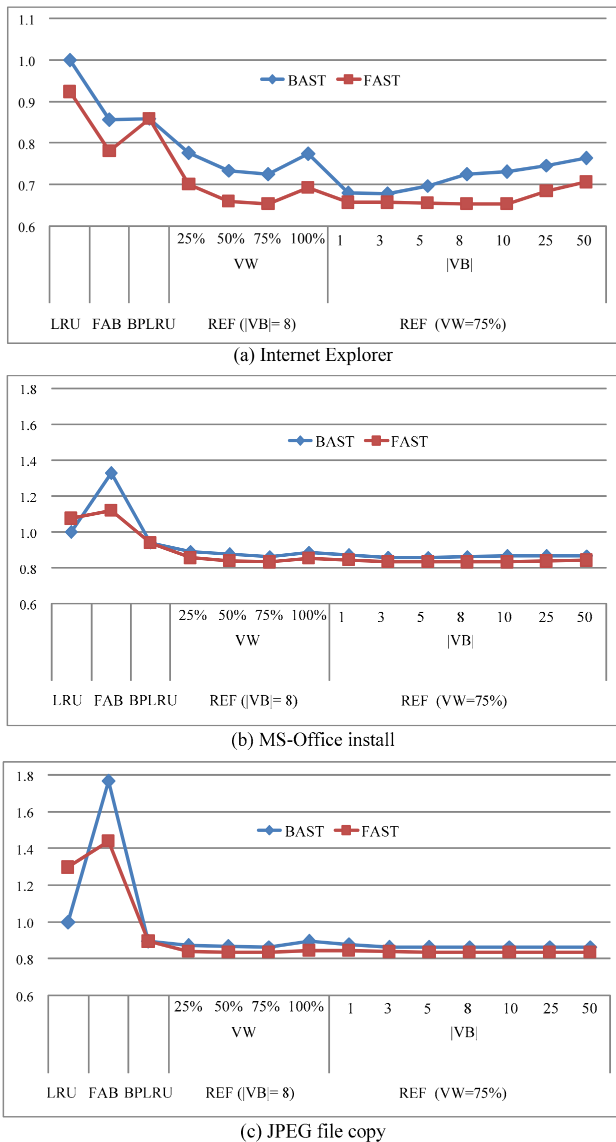
Structural parameter	Page size	2 KB
	Block size	128 KB (64 pages)
	Log buffer	8 blocks
	Buffer cache	16 MB
Timing Parameters (flash memory)	Page read	0.01 ms
	Page write	0.2 ms
	Block erase	2 ms

We compared the performances of LRU, FAB, and BPLRU schemes with that of REF scheme. For the REF scheme, we evaluated the performance varying the size of victim window (VW) with the fixed VB value of 8 and varying the number of victim blocks (VB) with the fixed VW value of 75%.

Fig. 8 shows the total execution times under several schemes. The execution times are normalized by the result when LRU buffer scheme and BAST FTL scheme are used. The total execution time means the time consumed in accessing flash memory. BPLRU shows poor performance for Internet Explorer trace. This is because the block padding overhead is large when the write pattern is significantly random. FAB shows poor performances for MS-Office install and JPEG copy applications. This is because FAB scheme ignores the recency of page thus sends the write requests on hot pages to the flash memory frequently. FAB is even worse than the LRU scheme.

The performances of REF are different depending on the sizes of VW and VB. When the size of VW is 75% and the number of VB is 3, the performance of REF is generally best. In Fig. 8(a), especially for the BAST scheme, the performance of REF is better as the number of VB is smaller. This is because the block thrashing is minimized. However, in the FAST scheme, the performances are similar when $|VB| \leq 8$ (the number of log blocks). REF improves the execution times by 20~30% over other schemes.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.



(a) Internet Explorer

(b) MS-Office install

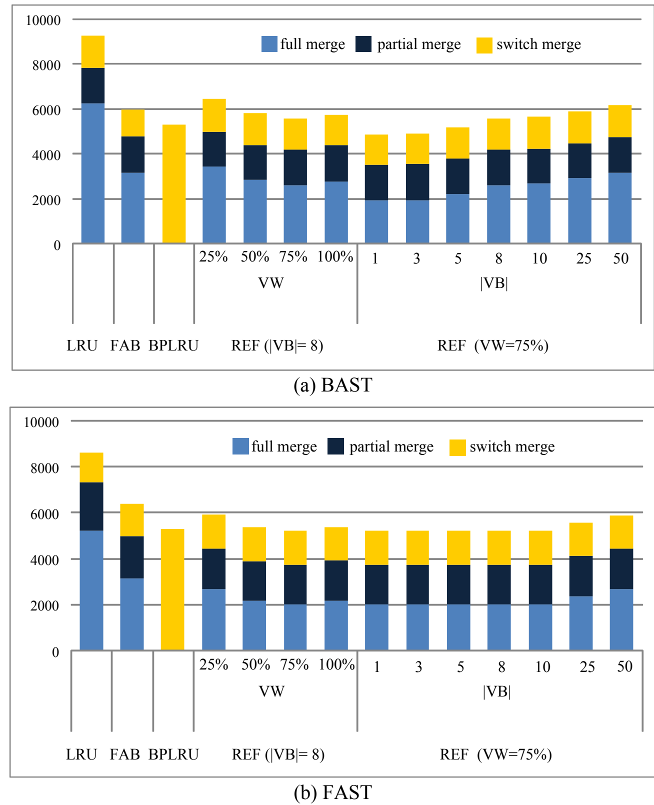
(c) JPEG file copy

Fig. 8. Comparisons of normalized execution times.

We also observed the behavior of FTL. Fig. 9 compares the numbers of block merges under several schemes. REF reduces especially the number of full merges compared to LRU and FAB schemes. BPLRU invokes only the switch merge. However, it requires a significant number of page copies.

Fig. 10 shows the numbers of page read, page write and block erase in flash memory. REF reduces both the numbers of reads and writes compared with LRU, FAB and BPLRU. Although BPLRU generates smaller numbers of block erase than REF, it requires large numbers of page read and write due to the block padding.

Finally, we evaluated the improvement of BP-REF over REF. Fig. 11 shows the execution times under BP-REF scheme normalized by those under REF scheme. We observed the performance gain of BP-REF over REF changing the block padding threshold. If the threshold is 100%, BP-REF is same to REF. If the threshold is 0%, BP-REF always performs the block padding. The performance gains of BP-REF are best when the threshold values are 10% and 40% in the BAST scheme and the FAST scheme, respectively. The performance gains are more significant for the FAST scheme compared to the BAST scheme. This is because the block padding eliminates the full merges which invoke high costs in the FAST scheme. As the threshold value is small, the number of switch merges increases and the numbers of full merges and partial merges decrease. From this result, we can know that the block padding threshold should be determined carefully considering the write pattern and the FTL scheme.



(a) BAST

(b) FAST

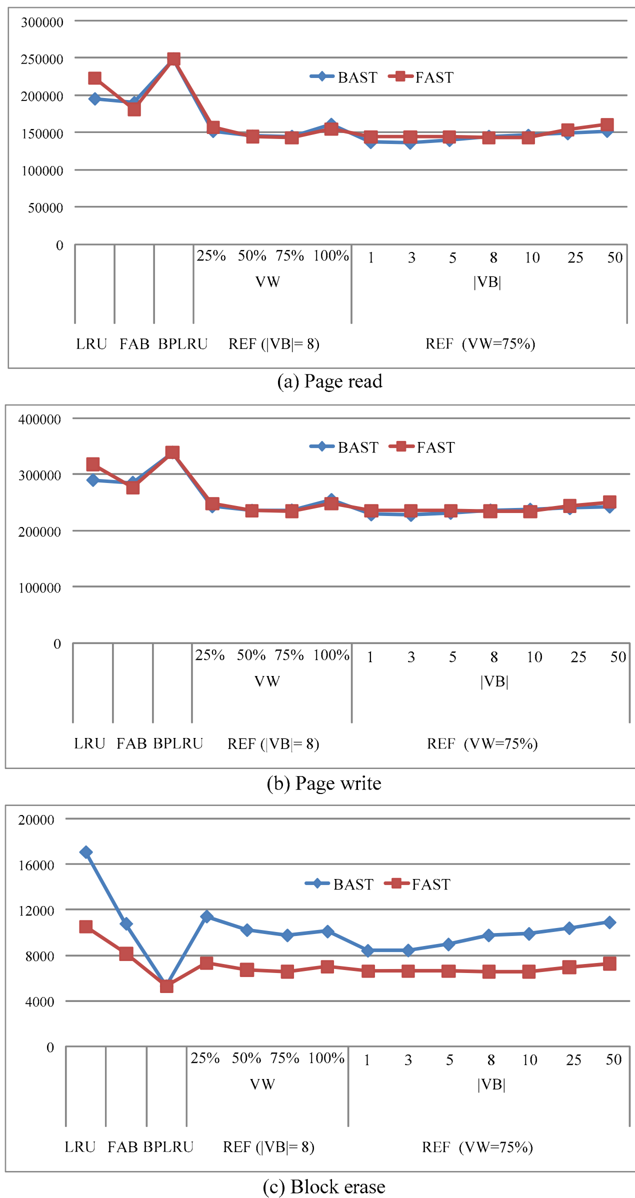
Fig. 9. Comparisons of merge count (Internet Explorer).

VI. Conclusion

We have presented a flash-aware page eviction technique called REF to solve the problems of log buffer-based FTLs and current flash-aware buffer management schemes. REF enforces the buffer cache to evict only the pages of the victim block. The victim block remains unchanged as far as possible to reduce the block thrashing and the block associativity. While FAB shows even worse performance than the LRU scheme depending on the input trace, REF shows better performances than LRU, FAB and BPLRU for all benchmarks. We also proposed the BP-REF scheme which performs block padding as well as flash-aware page eviction. Unlike the BPLRU scheme, the proposed BP-REF scheme does the block padding selectively to reduce the block padding overhead.

The proposed REF scheme can be further improved in several directions. For example, a dynamic adaptation technique is required, which adjusts the sizes of victim window and victim block observing the I/O pattern at run time. The block padding threshold also should be adjusted based on the I/O pattern.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.



(a) Page read

(b) Page write

(c) Block erase

Fig. 10. Comparisons of the number of NAND operations (Internet Explorer).

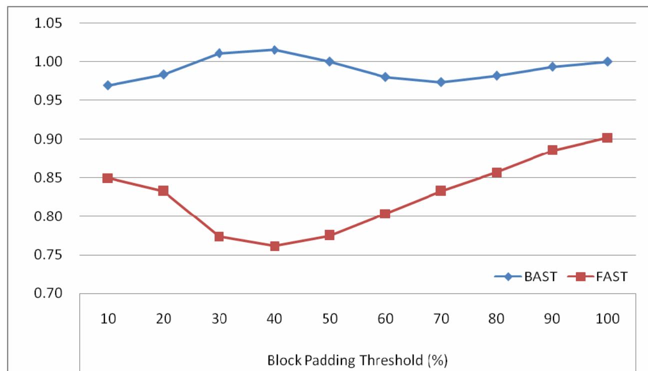


Fig. 11. Normalized execution times under BP-REF varying the block padding threshold (Internet Explorer).

REFERENCES

[1] Y.-J. Kim, S.-J. Lee, K. Zhang, and J. Kim, "I/O Performance optimization techniques for hybrid hard disk-based mobile consumer devices," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 4, pp. 1469–1476, 2007.

[2] J. Matthews, S. Trika, D. Hensgen, R. Coulson, and K. Grimsrud, "Intel Turbo Memory: Nonvolatile disk caches in the storage hierarchy of mainstream computer systems," *ACM Transactions on Storage*, vol. 4, no. 2, 2008.

[3] Y. H. Bae, "Design of a high performance flash memory-based solid state disk," *Journal of Korean Institute of Information Scientists and Engineers*, vol. 25, no. 6, 2007.

[4] J.-U. Kang, J. S. Kim, C. Park, H. Park, and J. Lee, "A multichannel architecture for high-performance NAND flash-based storage system," *Journal of Systems Architecture*, vol. 53, no. 9, pp. 644–658, 2007.

[5] 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 Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366–375, 2002.

[6] D. P. Bovet and M. Cesati, "Understanding the Linux Kernel," O'Reilly, 2006.

[7] 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 Transactions on Embedded Computing Systems*, vol. 6, no. 3, 2007.

[8] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "A superblock-based flash translation layer for nand flash memory," *Proc. of International Conference on Embedded Software*, pp. 161–170, 2006.

[9] S.-Y. Park, D. Jung, J.-U. Kang, J.-S. Kim, and J. Lee, "CFLRU: a replacement algorithm for flash memory," *Proc. of International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp. 234–241, 2006.

[10] H. Jo, J.-U. Kang, S.-Y. Park, J.-S. Kim, and J. Lee, "FAB: Flash-aware buffer management policy for portable media players," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 485–493, 2006.

[11] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," *Proc. of 6th USENIX Conference on File and Storage Technologies* (FAST), pp. 239–252, 2008.

Dongyoung Seo received the B.S. degree in computer engineering from Yeungnam University, Korea in 2003. Since 2003, he is an engineer of Samsung Electronics Co., Korea. He is also 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 in computer science and statistics, the M.S. degree in computer science, 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.

Authorized licensed use limited to: Sungkyunkwan University. Downloaded on June 08,2010 at 14:56:27 UTC from IEEE Xplore. Restrictions apply.