

# An Adaptive Partitioning Scheme for DRAM-based Cache in Solid State Drives

Hyotaek Shim<sup>†</sup>, Bon-Keun Seo<sup>†</sup>, Jin-Soo Kim<sup>‡</sup>, and Seungryoul Maeng<sup>†</sup>

<sup>†</sup>Computer Science Department, Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea  
{htshim, bkseo, maeng}@calab.kaist.ac.kr

<sup>‡</sup>School of Information and Communication Engineering, Sungkyunkwan University (SKKU), Republic of Korea  
jinsookim@skku.edu

**Abstract**—Recently, NAND flash-based Solid State Drives (SSDs) have been rapidly adopted in laptops, desktops, and server storage systems because their performance is superior to that of traditional magnetic disks. However, NAND flash memory has some limitations such as out-of-place updates, bulk erase operations, and a limited number of write operations. To alleviate these unfavorable characteristics, various techniques for improving internal software and hardware components have been devised. In particular, the internal device cache of SSDs has a significant impact on the performance. The device cache is used for two main purposes: to absorb frequent read/write requests and to store logical-to-physical address mapping information.

In the device cache, we observed that the optimal ratio of the data buffering and the address mapping space changes according to workload characteristics. To achieve optimal performance in SSDs, the device cache should be appropriately partitioned between the two main purposes. In this paper, we propose an *adaptive partitioning scheme*, which is based on a *ghost caching mechanism*, to adaptively tune the ratio of the buffering and the mapping space in the device cache according to the workload characteristics. The simulation results demonstrate that the performance of the proposed scheme approximates the best performance.

## I. INTRODUCTION

NAND flash-based storage devices, such as Solid State Drives (SSDs) [1], [2] or PCI-express flash cards [3], [4], have been widely used in laptops, desktops, and server storage systems because of their non-volatility, fast random access, shock resistance, small size, and low power consumption [5]. Moreover, SSDs are increasingly replacing Hard Disk Drives (HDDs), and the rate of this substitution is likely to increase as the price of NAND flash memory drops.

In spite of these advantages, NAND flash memory has an inherent limitation that its data must be erased in bulk before being overwritten, which is called *erase-before-write*. To hide this unfavorable difference with traditional storage devices, Flash Translation Layer (FTL) schemes have been devised. FTL provides a block device interface to the host by managing logical-to-physical address mapping information. The performance of SSDs heavily depends on FTL algorithms, but FTL involves a trade-off between the memory consumption for storing mapping information and the performance. If the

mapping granularity of FTL is more fine-grained, the FTL can achieve better performance, but it needs larger memory space.

SSDs usually adopt an internal device cache, such as DRAM or SDRAM, which is used for two purposes [6]–[8]. First, it provides data buffering to absorb frequent read and write requests. Second, it caches address mapping information for FTL. In existing studies, it is assumed that the ratio of the buffering and the mapping space (BM ratio) is fixed in the device cache, which we call a *static partitioning policy*. However, the optimal BM ratio is strongly affected by the characteristics of workloads such as the working set size, the ratio of read/write requests, and the degree of temporal locality. Therefore, the best performance can be achieved if we partition the device cache according to the workload characteristics. For example, a write-dominant workload with small working set size and high temporal locality may require larger buffering space (instead of larger mapping space), because most requests can be filtered by the data buffer. Accordingly, the fixed amount of the device cache must be appropriately partitioned between buffering versus mapping to improve the performance of SSDs.

In this paper, we propose an adaptive partitioning scheme that adaptively tunes the BM ratio in the device cache. For this scheme, we built a cost-benefit model based on a *ghost caching* [9], [10] mechanism. Through comparing the cost-benefit of increasing the buffering or the mapping space at every predefined interval, the device cache is adaptively partitioned. To prove the effectiveness of the proposed scheme, we implemented the proposed scheme with the two widely-used FTL algorithms: Demand-based Flash Translation Layer (DFTL) [11] and Fully Associative Sector Translation (FAST) [12]. The simulation results demonstrate that the performance of the proposed scheme approximates the most cost-beneficial balance between buffering and mapping. The proposed scheme enhanced the throughput by up to 41.9% more with 16MB DRAM when using the DFTL scheme, compared with that of the static partitioning policy.

## II. BACKGROUND AND RELATED WORKS

TABLE I  
SPECIFICATION OF NAND FLASH MEMORY  
(Samsung Electronics K9WAG08U1M [13], K9GAG08UXM [14])

Flash Type	Unit Size (KB)		Access Time ( $\mu$ s)		
	Page	Block	Read	Write	Erase
SLC	2	128	72.8	252.8	1500
MLC	4	512	165.6	905.6	1500

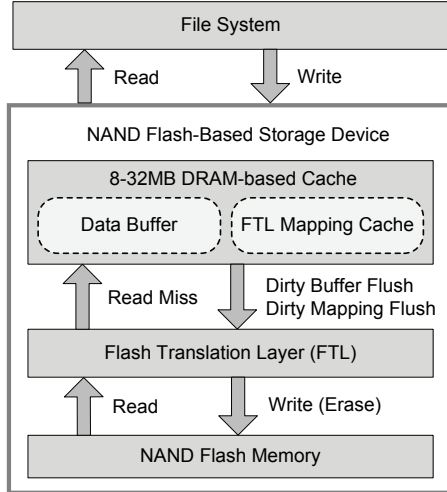


Fig. 1. Architecture of a typical solid state drive

### A. Characteristics of NAND Flash Memory

NAND flash memory is comprised of an array of blocks, each of which contains a fixed number of pages. NAND flash memory offers three basic operations: *read*, *write* (or *program*), and *erase*. A page is the unit of read and write operations, and a block is the unit of erase operations. An erase operation sets all data bits of a block to 1s. A read operation retrieves data from a page, while a write operation stores data on a page by changing some data bits to 0s.

There are two types of NAND flash memory. Single Level Cell (SLC) NAND [13] stores one bit per cell, whereas Multi Level Cell (MLC) NAND [14] provides two or more bits per cell for larger capacity. Table I shows the page/block sizes and the operation latencies of two representative NAND flash memory chips.

### B. Architecture of Solid State Drives

Fig. 1 illustrates the overall architecture of a typical NAND flash-based SSD. As previously mentioned, the DRAM-based device cache is partitioned into two major parts: the data buffering space and the FTL mapping space. When the file system generates read requests to the SSD, the buffer manager checks whether the requested data exists in the data buffer. If the data exists, the read request is handled in the data buffer, but otherwise is redirected to FTL.

When the file system writes data into the SSD, if the requested data is cached in the data buffer, the data can be simply overwritten. Otherwise, the buffer manager allocates an empty space in the buffer. If there is no more empty space, the buffer manager selects victim data in the buffer and flushes

the data to FTL. It is known to be more effective to use the data buffer only for write caching, not for read caching, because write latency is longer than read latency and writes involve erase operations in SSDs [8].

### C. Flash Translation Layer

The main role of FTL is to cover the idiosyncrasies of NAND flash memory so that SSD can emulate a traditional storage device that provides the block device interface. Through this compatibility layer, SSDs can be easily adopted in existing storage systems. To handle the erase-before-write feature of NAND flash memory, most FTLs assign write requests to previously-erased pages by keeping track of the logical-to-physical mapping information in an out-of-place manner. Thereafter, the pages that contain old data are invalidated. If there are no available previously-erased pages, FTL selects one or more victim blocks and triggers *garbage collection* in order to recycle them as free blocks. Before reclaiming the victim blocks, FTL should copy all valid pages in the victim blocks to free blocks reserved for garbage collection. We call this process *valid page migration* or *valid page copies*. After the valid page migration, the victim blocks are erased and eventually converted to free blocks. In the following subsections, we explain two representative FTL mapping schemes: page-level mapping and hybrid mapping.

1) *Page-Level Mapping Schemes*: In page-level mapping schemes, a Logical Page Number (LPN) is translated to a physical page number in NAND flash memory. Due to the flexibility in assigning empty pages, the best performance is accomplished even for random write patterns [15]. However, it requires a large memory footprint to maintain the fine-grained mapping information. Moreover, the size of mapping information increases in proportion to the capacity of SSDs. For example, when using 4 bytes for each page mapping entry with MLC NAND flash memory shown in Table I, we need 64MB of memory for 64GB of flash capacity. The large memory consumption is a major obstacle for large-capacity SSDs. To alleviate such a problem, the *Demand-based FTL scheme* (DFTL) [11] has been developed to create balance between the performance and the memory consumption. Basically, DFTL applies a caching mechanism to existing page-level mapping schemes.

DFTL maintains all logical-to-physical mapping information in the *translation pages* of NAND flash memory. A translation page consists of an array of mapping entries in sequential order in terms of LPN. All the mapping entries in each translation page have the same Virtual Translation Page Number (VPN) obtained through dividing their LPN by the maximum number of mapping entries within one page. Translation pages are written into translation blocks, which are separated from data blocks. Basically, to update logical-to-physical mapping, DFTL writes a new translation page that includes the new mapping entries after reading the existing translation page. Then, DFTL modifies the Global Translation Directory (GTD) that tracks all of the valid translation pages.

GTD is always maintained in the device cache since its size is very small.

DFTL keeps only frequently-accessed logical-to-physical mapping entries in the device cache, which is called the Cached Mapping Table (CMT). On a read or write request, if the related mapping entry exists in CMT, the request can be simply handled by updating CMT. For a read miss in CMT, DFTL inserts a new entry for the read request to CMT through reading the translation page to which the LPN of the request belongs. On a write miss, DFTL can create a new dirty entry for the write request without flash operations. If there is no empty space in CMT, the victim entries selected in LRU order are flushed to translation pages in NAND flash memory. To reduce flash writes for evicting dirty victim entries in CMT, DFTL flushes all of the dirty entries that belong to the same translation page at once, which is called a *batch update*. Compared with hybrid mapping schemes, which are explained later, the DFTL scheme achieves better performance with similar device memory consumption.

2) *Hybrid Mapping Schemes*: To balance the advantages of page-level and block-level mapping schemes, hybrid mapping schemes have been devised. Basically, such schemes are based on the block-level mapping scheme where all the pages in a block must be fully and sequentially written to preserve their relative offsets in the block. The hybrid mapping schemes offer block-level mapping for all *data blocks*, but they also support page-level mapping for a small fixed number of blocks called *log blocks* to handle write updates. Incoming write data is written in the log blocks incrementally from the first page. When all free blocks are consumed, FTL copies all valid pages within victim log blocks and their related data blocks into reserved free blocks. Then, the free blocks are remapped to new data blocks, while the victim log blocks and the old data blocks are erased and turned into free blocks.

Many hybrid mapping schemes have been developed [12], [16]–[18]. One of them, *Fully Associative Sector Translation* (FAST) [12], has been widely used in research and industrial areas. FAST uses two types of log blocks, RW and SW log blocks. The RW log blocks are used to handle random writes, while the SW log block is dedicated to accommodate sequential writes. FAST allocates only one SW log block for sequential writes, and all the other log blocks are used as RW log blocks. In FAST, all random updates for data blocks can be located in any RW log blocks.

When there are no more free RW log blocks, one of them is reclaimed in a *round-robin* fashion. To reclaim an RW log block, FAST should merge all associated data blocks that have valid pages in the RW log block. To merge an associated data block, FAST copies all valid pages that belong to the associated data block into a free block by searching all log blocks including the RW log block for a victim. After that, the free block becomes a new data block, and the associated data block is erased. This process is repeated until each associated data block is merged into new data blocks, which is called *full merges*, and the victim RW log block is finally erased. If there are enough RW log blocks and if the workload exhibits

high temporal locality for write requests, log pages in RW log blocks are likely to be invalidated soon by following writes. Accordingly, the more the number of invalidated pages in RW log blocks increases, the more the valid page migration is mitigated.

#### D. Power Failure Recovery

Buffered data and mapping information that are maintained in the volatile device cache can be lost by unexpected power failures. Simple approaches to prevent the loss of the important data in the device cache are to employ either (1) non-volatile memory devices [19] such as phase change RAM (PRAM) [20] and ferroelectric RAM (FRAM) [21], (2) traditional battery-backed DRAMs, or (3) a super cap that provides enough power to flush all of the dirty data in the device cache to NAND flash memory.

The Lightweight Time-shift Flash Translation Layer (LTFTL) is an example of software-based approach that aims at maintaining FTL consistency in case of abnormal shutdown [22]. In this scheme, FTL maintains previous data pages in the log area without reclaiming them until a checkpoint. After detecting the abnormal shutdown at initialization, LTFTL turns back to the previous state of time periods, which has consistency, by changing the mapping of data pages in the log area to their previous data pages. This technique is well suited for the *out-of-place update* property of NAND flash memory.

#### E. Related Works

The idea of this paper was inspired by some previous studies: Adaptive Replacement Cache (ARC) [9] and Patterson’s work [10]. ARC is a cache management algorithm to increase the hit ratio in the storage cache. According to workload characteristics, this scheme adaptively balances between recency and frequency by using a learning rule to track a workload. ARC works uniformly well with varied workloads and cache size without workload-specific tuning or prior knowledge. Patterson et. al. proposed proactive file cache mechanisms to reduce seek and rotational latencies and to utilize I/O parallelism in an array of hard disk drives. This study focused on dynamically balancing caching against prefetching by exploiting hints about future I/O demands from user applications. In this paper, we adopted the concept of adaptively tuning into the buffer cache of SSDs through considering the characteristics of NAND flash memory and FTL algorithms.

### III. ADAPTIVE PARTITIONING SCHEME

#### A. Motivation

If we allocate more buffering space to the device cache, better performance can be achieved due to the reduced miss ratio of read and write requests. The performance of SSD can also benefit from the increased mapping space, since it will lower the number of flash operations needed to access the mapping information. Therefore, with a fixed amount of the device cache in SSD, a trade-off should be made between how much space is allocated to buffering versus mapping. The

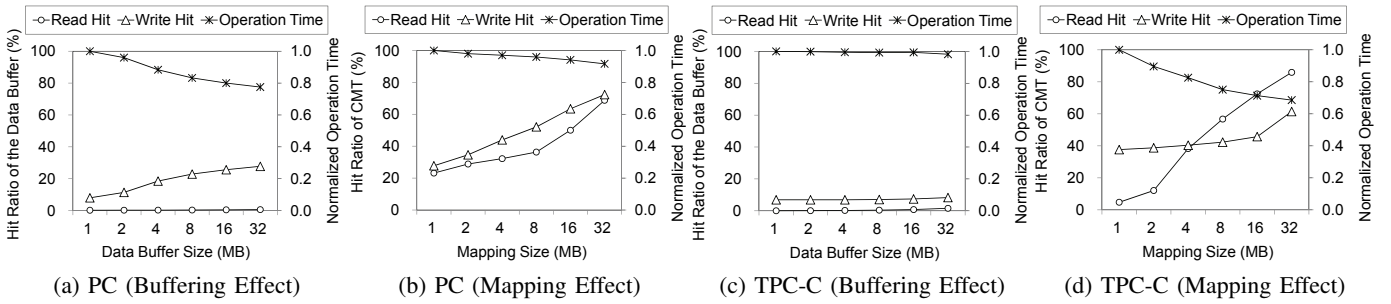


Fig. 2. Effects of increasing the buffering or the mapping space

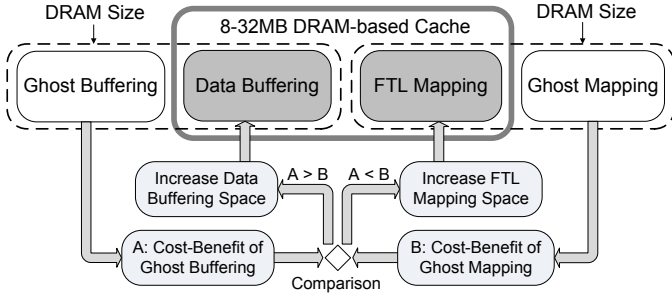


Fig. 3. Overall structure of the adaptive partitioning scheme

problem is that the optimal BM ratio is usually affected by workload characteristics.

To show this more clearly, we measured the total operation time by varying the buffering or the mapping space from 1MB to 32MB, while fixing the other space to 1MB. Specific information about the simulation configurations and the used traces is explained in Section IV. Fig. 2(a) and (b) show the total operation time with the PC trace under the DFTL scheme. In this workload, increasing the buffer space is more effective in enhancing the performance than increasing the mapping space. Since this workload has small working set size and a high write hit ratio, the larger data buffer can absorb more writes, thus also removing FTL operations. In the TPC-C trace shown in Fig. 2(c) and (d), increasing the mapping space is more effective in reducing the operation time, because this workload has large working set size. Under this workload, the buffering space must be quite larger to capture frequent requests enough. These results show that we need to adjust the size of the buffering and the mapping space adaptively according to the characteristics of target workloads to obtain the best overall performance.

In order to maximize the performance of the device cache, we should store the most cost-beneficial data that provides the largest performance benefit per memory consumption. Since this problem is mapped to a *fractional knapsack problem*, including the problem to determine the appropriate BM ratio, we can exploit a greedy approximation to insert more cost-beneficial data into the device cache. The key method of our approach is to dynamically adjust the ratio through comparing the cost-benefits of buffering and mapping at regular intervals according to the workload characteristics.

## B. Main Idea

Fig. 3 shows the main idea of the proposed adaptive partitioning scheme. In our scheme, we maintain the ghost buffer and the ghost mapping caches, which are a kind of *exclusive victim cache*, that store only metadata without actual data. When a victim data is flushed from the data buffer or the mapping cache, the metadata of the victim is inserted into the ghost buffer or the ghost mapping cache. Through these ghost caches, we estimate the cost-benefit of their actual caches. Specifically, it is assumed that if the actual caches have more space, they will obtain profits as much as their ghost caches provide. In this way, we expect that it is more beneficial to increase the size of the actual cache whose ghost cache provides a larger cost-benefit value than the other ghost cache.

We estimate the profit of ghost caches as an opportunity cost that means all the costs (NAND flash operation time) caused by not enlarging the actual cache size. The opportunity cost is generated by read or write misses in the actual caches that correspond to all read or write hits in the ghost caches; when a read or write hit for a request occurs in the ghost caches, a read or write miss for the same request also occurs in the actual caches. Whenever read or write hits occur in the ghost caches, we compute the opportunity cost caused by the corresponding read or write misses in the actual caches to appraise the profits of ghost caches. At every pre-defined interval, we calculate the cost-benefit of the ghost caches, and we tune the BM ratio by comparing the cost-benefit values of their ghost caches.

In the proposed scheme, the maximum size of an actual cache and its ghost cache is set to the total size of the DRAM-based device cache, since the size of the buffering or the mapping space cannot be larger than the device cache size. Note that the size of a ghost cache reflects its actual data size although the ghost cache stores only metadata. We also assume that the expected memory consumption needed for caching the actual data of ghost caches is the *cost* of the cost-benefit value.

## C. Adaptive Partitioning Algorithm

We set up a model to calculate the cost-benefit values of ghost caches ( $C_{GB}$  and  $C_{GM}$ ). The parameters used in the model are summarized in Table II. Based on the cost-benefit model, we calculate the profit of ghost mapping ( $P_{GM}$ ) and

TABLE II

MODEL PARAMETERS TO ANALYZE THE COST-BENEFIT OF GHOST CACHES

Notation	Description
$N_{req}$	Request number
$N_{intv}$	Interval for applying the adaptive partitioning scheme
$N_{r\_GB}, N_{w\_GB}$	Hit counts of page reads and writes in the ghost buffer cache
$N_{r\_GM}, N_{w\_GM}$	Hit counts of page reads and writes in the ghost mapping cache
$PF_{r\_GB}, PF_{w\_GB}$	Profit factors of a read and a write hit in the ghost buffer cache
$PF_{r\_GM}, PF_{w\_GM}$	Profit factors of a read and a write hit in the ghost mapping cache
$S_{GB}$	Current size of the ghost buffer cache
$S_{GM}$	Current size of the ghost mapping cache

that of ghost buffering ( $P_{GB}$ ) as follows:

$$\begin{aligned}
P_{GM} &= P_{r\_GM} + P_{w\_GM} \\
P_{r\_GM} &= N_{r\_GM} * PF_{r\_GM} \\
P_{w\_GM} &= N_{w\_GM} * PF_{w\_GM} \\
P_{GB} &= P_{r\_GB} + P_{w\_GB} \\
P_{r\_GB} &= N_{r\_GB} * PF_{r\_GB} \\
P_{w\_GB} &= N_{w\_GB} * PF_{w\_GB}
\end{aligned}$$

First, we explain how to get  $P_{GM}$ , which is obtained by summing up the profits caused by read and write hits in the ghost mapping cache ( $P_{r\_GM}$  and  $P_{w\_GM}$ ). The profit is calculated by opportunity costs generated from the corresponding read and write misses in the actual mapping cache. At every read hit in the ghost mapping cache ( $N_{r\_GM}$ ), we increase  $P_{r\_GM}$  by the opportunity cost ( $PF_{r\_GM}$ ) of a read miss in the actual mapping cache. For example, on a read miss in the actual mapping cache, FTL should read the related mapping information from NAND flash memory, and thus we consider this overhead as an opportunity cost for a read miss in the actual mapping cache and also as a profit for a read hit in the ghost mapping cache.  $P_{w\_GM}$  can be similarly estimated from the opportunity cost ( $PF_{w\_GM}$ ) caused by write misses in the actual mapping cache. The write misses are restricted to those that correspond to write hits ( $N_{w\_GM}$ ) in its ghost mapping cache.

Second,  $P_{GB}$  consists of the profits caused by read and write hits in the ghost buffer ( $P_{r\_GB}$  and  $P_{w\_GB}$ ). To calculate the profits, we should appraise opportunity costs caused by read and write misses in the actual buffer. For every read or write hit in the ghost buffer, we increase  $P_{GB}$  by the estimated read or write opportunity cost ( $PF_{r\_GB}$  or  $PF_{w\_GB}$ ).

Algorithm 1 shows the pseudo-code of the adaptive partitioning scheme. We obtain the cost-benefit value of increasing the buffering space ( $C_{GB}$ ) by means of dividing the profit of the ghost buffer ( $P_{GB}$ ) by the current size of the ghost buffer ( $S_{GB}$ ). The cost-benefit value of increasing the mapping space ( $C_{GM}$ ) is also calculated similarly. At every interval ( $N_{intv}$ ), we compare the cost-benefit values of ghost buffering and ghost mapping. If the cost-benefit value of ghost buffering

---

**Algorithm 1** Adaptive Partitioning Algorithm

---

```

1: procedure ADAPTIVE_PARTITIONING()
2:   if ( $N_{req} \bmod N_{intv} == 0$ ) then
3:      $C_{GB} = P_{GB}/S_{GB}$ 
4:      $C_{GM} = P_{GM}/S_{GM}$ 
5:     if ( $C_{GB} < C_{GM}$ ) then
6:       INCREASE_MAPPING_SPACE( $C_{GM}/C_{GB}$ )
7:     else if ( $C_{GB} > C_{GM}$ ) then
8:       INCREASE_BUFFER_SPACE( $C_{GB}/C_{GM}$ )
9:     end if
10:     $N_{r\_GB}, N_{w\_GB}, N_{r\_GM}, N_{w\_GM} = 0$ 
11:   end if
12: end procedure
13:
14: procedure INCREASE_MAPPING_SPACE( $inc\_count$ )
15:    $S_{tune} = S_{tune\_unit} \times inc\_count$ 
16:    $S_{max\_map} = S_{max\_map} + S_{tune}$ 
17:    $S_{max\_buf} = S_{max\_buf} - S_{tune}$ 
18:
19:   while ( $S_{max\_buf} < S_{cur\_buf}$ ) do
20:     flush_victim_buffer()
21:   end while
22: end procedure
23:
24: procedure INCREASE_BUFFER_SPACE( $inc\_count$ )
25:    $S_{tune} = S_{tune\_unit} \times inc\_count$ 
26:    $S_{max\_buf} = S_{max\_buf} + S_{tune}$ 
27:    $S_{max\_map} = S_{max\_map} - S_{tune}$ 
28:
29:   while ( $S_{max\_map} < S_{cur\_map}$ ) do
30:     decrease_mapping_space()
31:   end while
32: end procedure

```

---

is larger than that of ghost mapping, we increase the buffering space by the tuning size ( $S_{tune\_unit}$ ) multiplied by the cost-benefit factor ( $C_{GB}/C_{GM}$ ), and we decrease the mapping space by the same amount. After adjusting the BM ratio, we clear the read and the write hit counts in the ghost caches for the next period.

We apply this adaptive partitioning scheme to two widely-used FTLs: DFTL and FAST schemes. In the following subsections, we explain how to concrete the adaptive partitioning algorithm for each FTL algorithm. Specific notations for NAND flash memory, DFTL, and FAST are defined in Table III. For data buffer management, we use a page-level LRU buffer replacement algorithm.

#### D. Case Study for Demand-based FTL (DFTL)

DFTL is a caching-based FTL scheme similar to the superblock FTL [17] and  $\mu$ -FTL [18] schemes, where only frequently accessed mapping information is stored into the mapping cache. As the capacity of SSDs considerably increases, the amount of FTL mapping information will also increase, and thus such caching-based FTL schemes will

TABLE IV  
INSTANCES OF MODEL PARAMETERS TO APPLY THE ADAPTIVE PARTITIONING SCHEME INTO DFTL AND FAST

Notation	DFTL	FAST
$N_{r\_GM}$	$N_{r\_GCMT}$	None
$N_{w\_GM}$	$N_{w\_GCMT}$	$N_{w\_Glog}$
$PF_{r\_GM}$	$T_r$	None
$PF_{w\_GM}$	$(T_r + T_w + T_{two}) / F_b$	$T_{full\_merge} / F_{as\_set}$
$PF_{r\_GB}$	$T_r + T_r * R_{r\_mis\_CMT}$	$T_r$
$PF_{w\_GB}$	$T_w + T_{two} + (T_r + T_w + T_{two}) / F_b * R_{w\_mis\_CMT}$	$T_w + T_{log\_GC} / N_{ppb}$
$S_{tune\_unit}$	$S_{CMT\_entry}$	$S_{log\_entry}$

TABLE III  
SPECIFIC NOTATIONS TO CALCULATE THE PROFIT OF THE GHOST CACHES

Notation	Description
NAND Flash Memory	
$N_{ppb}$	Number of pages per block
$T_r, T_w, T_e$	Read, write, and erase operation time
DFTL	
$N_{r\_GCMT}$	Hit count of entry reads in the ghost CMT
$N_{w\_GCMT}$	Hit count of entry writes in the ghost CMT
$T_{trans\_GC}$	Average cost for reclaiming a translation block
$T_{data\_GC}$	Average cost for reclaiming a data block
$T_{two}$	Overhead for writing a translation page: $T_{trans\_GC} / N_{ppb}$
$T_{dwo}$	Overhead for writing a data page: $T_{data\_GC} / N_{ppb}$
$R_{r\_mis\_CMT}$	Read miss ratio in the actual CMT
$R_{w\_mis\_CMT}$	Write miss ratio in the actual CMT
$F_b$	Average number of CMT entries flushed by a batch update
$S_{CMT\_entry}$	Memory size needed for one entry in CMT
FAST	
$N_{w\_Glog}$	Hit count of page writes in the ghost log blocks
$F_{as\_set}$	Average number of written pages that belong to the same associated data block for each victim RW log block
$T_{full\_merge}$	Full merge cost for an associated data block in RW log blocks: $N_{ppb} * (T_r + T_w) + T_e$
$T_{log\_GC}$	Average cost for reclaiming one RW log block
$S_{log\_entry}$	Memory size of mapping information for one log block

dominate because of the limited capacity of the device cache. Those FTLs are good targets for the proposed scheme since their performance heavily depends on the size of the mapping cache. We apply the proposed adaptive partitioning scheme to DFTL by concreting the profit factors ( $PF_{r\_GM}$ ,  $PF_{w\_GM}$ ,  $PF_{r\_GB}$ , and  $PF_{w\_GB}$ ) and the tuning size ( $S_{tune\_unit}$ ) shown in Table IV. Based on the cost-benefit model mentioned in the previous subsection, we calculate the profit factors of ghost mapping and ghost buffering.

We explain how to get the read profit factor of the ghost CMT ( $PF_{r\_GM}$ ). On a read miss in the actual CMT, DFTL should read one translation page from NAND flash memory, and thus we consider one-page read latency ( $T_r$ ) as the opportunity cost ( $PF_{r\_GM}$ ) for a read miss in the actual CMT and also as the profit for a read hit in the ghost CMT.

For  $PF_{w\_GM}$ , we should estimate the opportunity cost caused by a write miss in the actual CMT. On a write miss in the actual CMT, a dirty entry can be simply created without any flash operation, but to flush the dirty entry subsequently, the corresponding translation page should be read ( $T_r$ ) and

updated ( $T_w$ ), including write overheads ( $T_{two}$ ) for reclaiming translation blocks. Additionally, considering the batch update technique that flushes all dirty entries that belong to the same translation page at once, we divide the flushing cost by a batch factor ( $F_b$ ). The batch factor means the average number of CMT entries flushed by writing one translation page into NAND flash memory. Consequently, the profit factor of a write hit in the ghost CMT is calculated as  $(T_r + T_w + T_{two}) / F_b$ .

The profit factors of the ghost buffer ( $PF_{r\_GB}$  and  $PF_{w\_GB}$ ) consist of profits caused by a read hit and a write hit in the ghost buffer. To calculate these profit factors, we should estimate the opportunity cost of a read and a write miss in the actual buffer. The opportunity cost of a read miss in the actual buffer consists of  $T_r$  for the missed data page in NAND flash memory and  $T_r * R_{r\_mis\_CMT}$  for reading the missed translation page in the case of a read miss in CMT. The opportunity cost of a write miss in the actual CMT is comprised of  $T_w$  for writing the missed data page, average GC overheads for a write ( $T_{dwo}$ ), and  $(T_r + T_w + T_{two}) / F_b * R_{w\_mis\_CMT}$  for inserting a new dirty entry into CMT if a write to CMT is missed.

In DFTL, the tuning size of the proposed scheme is the memory size needed for one CMT entry ( $S_{CMT\_entry}$ ). To reduce the mapping space, DFTL can easily reclaim the memory space allocated for CMT by removing clean entries or by flushing dirty entries in CMT. The proposed scheme can also increase the mapping space after decreasing the buffering space by flushing victim pages in the data buffer.

#### E. Case Study for Fully Associative Sector Translation (FAST)

As one of other log block-based FTLs, the FAST scheme maintains page-level mapping information for all extra blocks in the mapping cache. For write-dominant workloads that have large working set size and high temporal locality, the performance is significantly improved as the number of log blocks increases. This is because many valid pages in log blocks can be invalidated by the following updates, which are *write hits*, reducing the overhead of valid page migration. For other workloads that have small working set size or read-dominant requests, however, a large number of log blocks are likely to remain unused, unnecessarily wasting a large portion of the device cache for maintaining mapping information for those log blocks. If SSD utilizes the unused mapping space for data buffering by reclaiming the mapping space for surplus log blocks, it can increase the read and write hit ratios in the data buffer. Hence, the FAST scheme can be another target for

TABLE V  
SUMMARY OF THE BLOCK-LEVEL TRACES THAT MODEL VARIOUS WORKLOADS

Name	Description	Avg. Request Size [Read/Write] (KB)	Request Ratio [Read/Write] (%)	Working Set Size [Read/Write] (GB)
SYSmrk	Running SYSmrk 2007 Preview including e-learning, office works, video creation, and 3D modeling	13.6 / 20	33 / 67	0.11 / 0.24
PC	Document-based realistic workloads using various office applications	20 / 13.4	23.7 / 76.3	5.82 / 8.45
Financial	I/O trace from an OnLine Transaction Processing (OLTP) application running at a financial institution	2.3 / 3.6	47.4 / 52.6	0.45 / 0.5
TPC-C	Running a TPC-C benchmark test with <i>Benchmark Factory</i>	2.2 / 2.1	81.4 / 18.6	8.04 / 4.45

the proposed scheme. Similar to the approach in the previous subsection, we explain a specific cost-benefit model of the adaptive partitioning scheme for FAST as shown in Table IV.

In FAST, the profit factors of ghost mapping ( $PF_{r\_GM}$  and  $PF_{w\_GM}$ ) are obtained by considering the opportunity cost caused by insufficient log blocks. Since flash read operations have the same cost regardless of the number of log blocks, there is no opportunity cost for read operations. In other words, there is no read miss in the mapping cache since block-level mapping information is always kept in the device cache. Accordingly, there are no corresponding instances for  $N_{r\_GM}$  and  $PF_{r\_GM}$ .

For flash write operations, FAST can reduce full merges to reclaim RW log blocks if it has more log blocks. For example, when all valid pages that belong to the same associated data block in an RW log block are invalidated by the following updates, FAST can avoid a full merge for the associated data block when reclaiming the RW log block. From this point of view, the proposed scheme maintains the metadata information about recently-migrated valid pages from victim RW log blocks in the ghost log blocks. The opportunity cost for a write miss that corresponds to a write hit in the ghost log blocks is calculated as  $T_{full\_merge}/F_{as\_set}$ .  $F_{as\_set}$  means the average number of written pages that belong to the same associated data block for each victim RW log block.

The profit factors of the ghost buffering ( $PF_{r\_GB}$ ,  $PF_{w\_GB}$ ) are calculated by the profits of read and write hits in the ghost buffer. The opportunity cost ( $PF_{r\_GB}$ ) for a read miss in the actual buffer is simply  $T_r$  for reading the data page from NAND flash memory. The opportunity cost ( $PF_{w\_GB}$ ) for a write miss in the actual buffer is comprised of  $T_w$  for writing the missed page and  $T_{log\_GC}/N_{ppb}$  for one-page write overheads in NAND flash memory.

In FAST, the tuning size of the proposed scheme is the memory size of mapping information needed for one log block ( $S_{log\_entry}$ ). To reduce the mapping space, FAST empties the memory space allocated for log blocks by decreasing the number of free log blocks or by reclaiming victim log blocks.

#### IV. PERFORMANCE EVALUATION

##### A. Methodology

To evaluate the performance of the proposed scheme, we implemented a trace-driven simulator. The capacity of NAND flash memory was configured as 64GB SLC NAND flash memory [14], whose specification is described in Table I. SLC

NAND flash memory is widely used in desktop and server storage systems. We assumed 8MB and 16MB of DRAM as an internal device cache, and the data buffer is used only for write caching, not for read caching. We configured the tuning interval as 1000 and 10000 requests for the proposed scheme with DFTL and FAST, respectively, considering that the tuning size in FAST is larger than that in DFTL.

The metadata in the ghost buffer is replaced with a page-level LRU policy, which is the same as the replacement algorithm of the actual buffer. To implement ghost mapping, we used a *bloom filter* to count the read and write hits in the ghost mapping. To insert an LPN into the ghost mapping cache, we set the bit of the corresponding hash bucket, and we reset the bit in random order to flush the victim of the ghost mapping cache. If hash collisions occur in the hash bitmap when LPNs are inserted, the bit count of hash bitmap does not increase, but can be rather getting smaller by flushing. To avoid this, the proposed scheme conditionally flushes the hash bitmap when the ghost mapping space needs to be reduced. If (the current bit count of the hash bitmap)  $\times \alpha$  is smaller than the bit count that should be preserved without collisions, the proposed scheme decreases the only size value of the ghost mapping cache without flushing the hash bitmap. In our simulation, the  $\alpha$  is configured as 3, and the hash bitmap for ghost mapping is maintained in the device cache (DRAM) since it occupies only 128KB. In addition, for the proposed scheme, the metadata of the ghost buffer is also stored in the data cache.

Table V summarizes the characteristics of the four traces used in this paper. They were collected by block-level tracing tools, *DiskMon* [23] and *blktrace* [24]. The four traces are used to evaluate the performance of the proposed techniques in various environments of desktop and server storage systems. To evaluate the performance in a typical personal computer environment, we use the trace of a well-known PC benchmark called SYSmrk. In addition, the PC trace obtained from a real user is used to model the realistic workload of a desktop PC. The Financial and TPC-C traces represent the workloads primarily used in server systems.

##### B. Operation Time with DFTL

In this section, we evaluate the performance enhancement of the adaptive partitioning scheme for the DFTL scheme. Under the different workloads shown in Table V, we measured the total operation time in NAND flash memory with the different



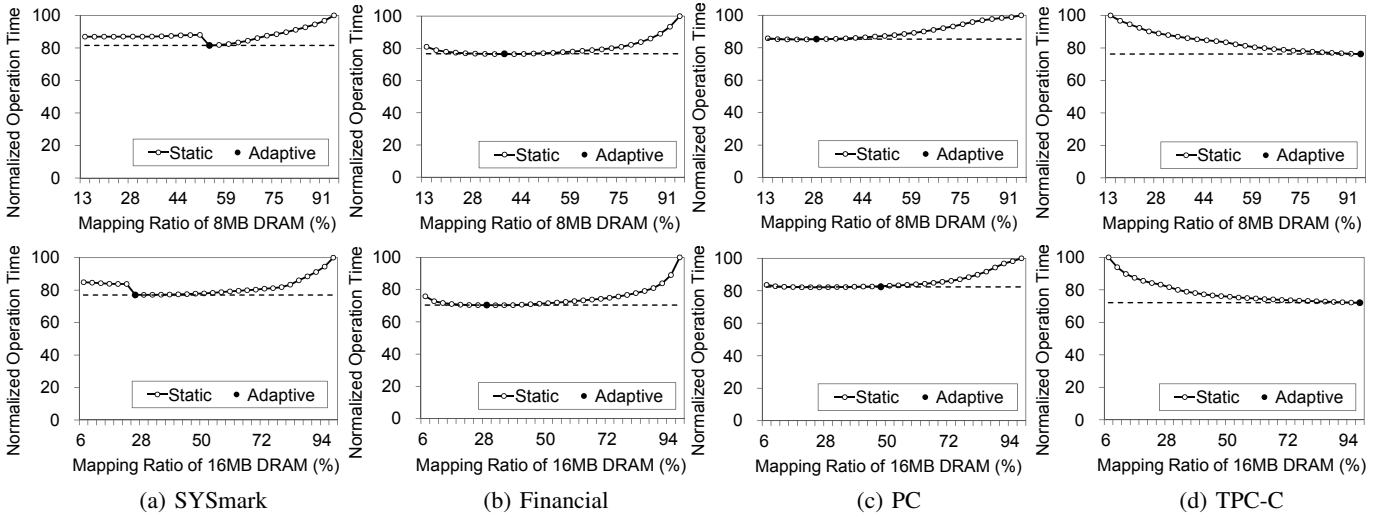


Fig. 4. Operation time comparison under various workloads with DFTL

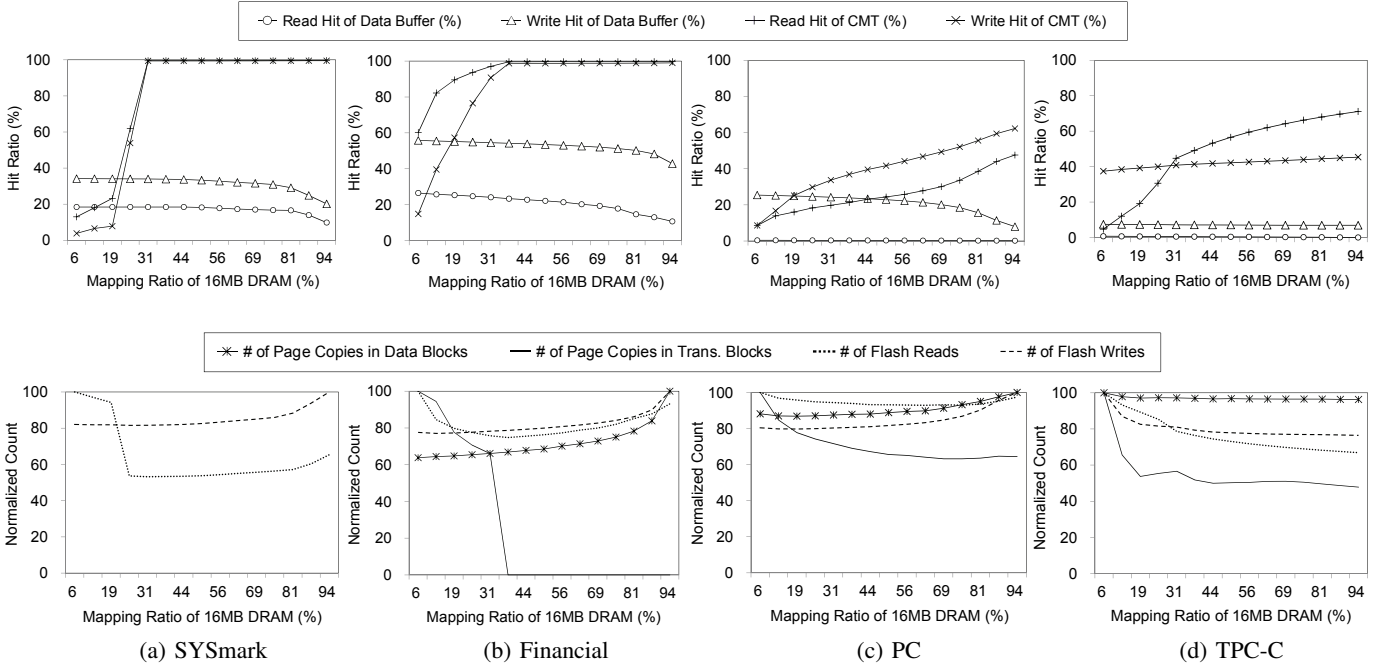


Fig. 5. Effects of the BM ratio with DFTL

device cache sizes, 8MB and 16MB. The proposed scheme is compared with the static partitioning policy where the BM ratio is fixed during the simulation. For the *static partitioning policy*, we measured the performance several times under different ratios from 0% to 100% of the device cache. For the proposed scheme, we configured the initial ratio of the mapping space as 50% of the device cache, and the ratio is adjusted autonomously between 0% and 100% during the simulation. Finally, the number of extra blocks, which are used for storing FTL mapping information and writing data, was set as 3% of the total flash capacity.

Fig. 4 compares the operation time of each trace, which is normalized to the longest one. In the graphs, the result of the adaptive partitioning scheme is displayed as a point, whose

x-axis position means the average ratio of the mapping space for all tuning periods. From this result, we observe that the performance significantly changes according to the BM ratio in the device cache. Moreover, the optimal ratio for the best performance is determined by the workload characteristics. To analyze the effect caused by the BM ratio more specifically, we measured detailed parameters with 16MB of DRAM under the static partitioning policy as shown in Fig. 5. The read and write hit ratios in the data buffer and CMT reflect the trade-offs between mapping and buffering. In addition, we inspect the concrete effects of adjusting the mapping space against the buffering space through the number of valid page copies in data and translation blocks, and the number of flash read and write operations. These counts are normalized to their largest



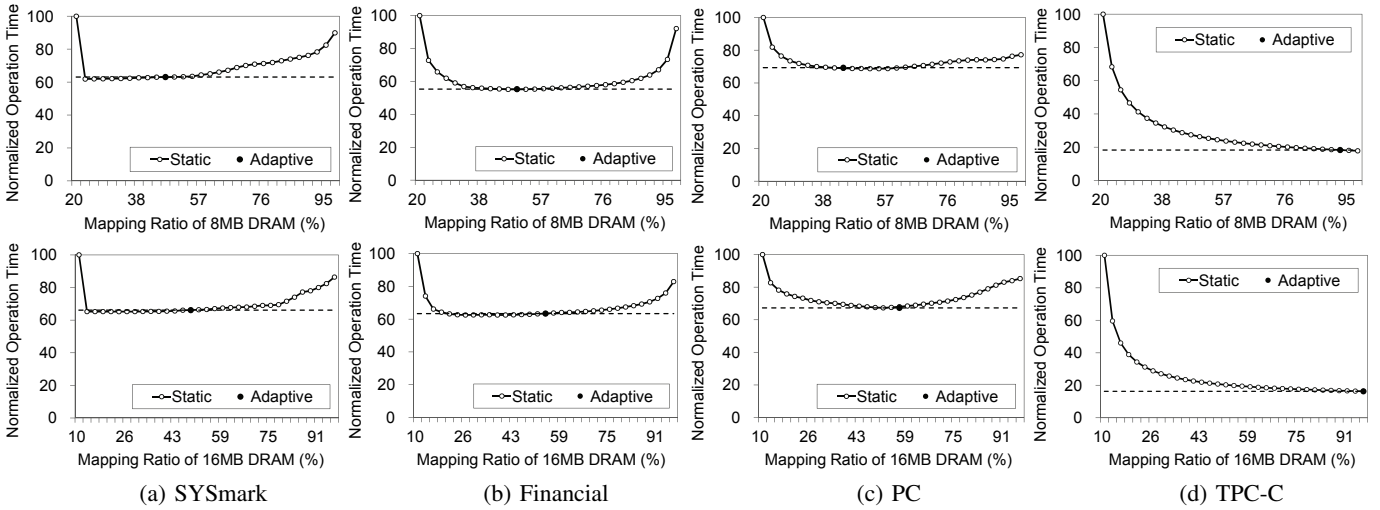


Fig. 6. Operation time comparison under various workloads with FAST

counts, respectively.

In the results of SYSmark and Financial shown in Fig. 4(a) and (b), the best performance is achieved when the ratio of mapping space is about 30% or 50%. As the ratio of mapping space increases from zero to around more than 30%, Fig. 5(a) and (b) show that the read and write hit ratios in CMT rapidly reach almost 100%, because these workloads exhibit small working set size and high temporal locality. Note that the mapping space can accommodate more information and provide better profits with the same memory size, compared with the data buffer. From the point of view of the buffering space, the read and write hit ratios in the data buffer are slightly decreased as the ratio of buffering space decreases from 100% to around 70%. Accordingly, the read and write hit ratios in CMT can greatly increase by consuming only a small amount of DRAM without sacrificing the hit ratios in the data buffer.

This increment of read and write hit ratios in CMT reduces the number of flash read operations, thus reducing the operation time. As the mapping space increases more than the working set size, however, the reduced buffer space increasingly decreases the read and write hit ratios of the data buffer. In addition, there is no profit from the additional mapping space since this space remains unused. In particular, when the ratio of buffering space is less than 30%, the hit ratios in the data buffer sharply drop. As a result, read and write operations are more generated in NAND flash memory.

In the PC trace shown in Fig. 4(c) and Fig. 5(c), as the mapping space increases, the read and write hit ratios of CMT are gently raised, thus reducing the number of flash read operations and valid page copies in translation blocks. At the same time, however, the number of valid page copies in data blocks increases due to the reduced write hit ratio in the data buffer. Consequently, the best performance is achieved at around 25%, which is the ratio that provides the largest cost-benefit between buffering and mapping.

The TPC-C trace in Fig. 4(d) requires large mapping space

for the best performance, compared with the other workloads that need a small amount of the device cache for the mapping space. The TPC-C trace exhibits large working set size, low temporal locality, and read-dominant workloads. Accordingly, increasing the buffer space has a little effect on the overall performance. However, increasing the mapping space has a significant effect on increasing the read hit ratio in CMT and mitigating valid page migration in translation blocks. In particular, the flash reads are significantly reduced as the mapping space reaches almost 100%.

For all the traces, we find that the performance of the adaptive partitioning scheme approximates the best performance very closely, since the average BM ratio of the proposed scheme dynamically tracks the optimal BM ratio well under the various workload characteristics. Therefore, we demonstrate that the proposed scheme achieves the best performance with DFTL in different environments.

### C. Operation Time with FAST

We also measured the operation time and additional performance factors for FAST. Fig. 6 presents the operation time according to the ratio of the mapping space with the different DRAM sizes, 8MB and 16MB. As mentioned in the previous subsection, in these graphs in Fig. 6, the result of the adaptive partitioning scheme is displayed as a point, whose x-axis position means the average ratio of the mapping space. In FAST, the mapping space is used to store mapping information for data blocks and log blocks. As the size of the mapping space enlarges, the number of log blocks can increase, but is limited by the number of extra blocks. In the following simulations with FAST, the number of extra blocks is configured up to about 10% of the total flash capacity, considering the specification of the existing SSDs [2].

Similar to the previous subsection, for the *static partitioning policy*, we measured the performance several times under different mapping ratios from 0% to 100% of the device cache. For the adaptive partitioning scheme, we configured the initial

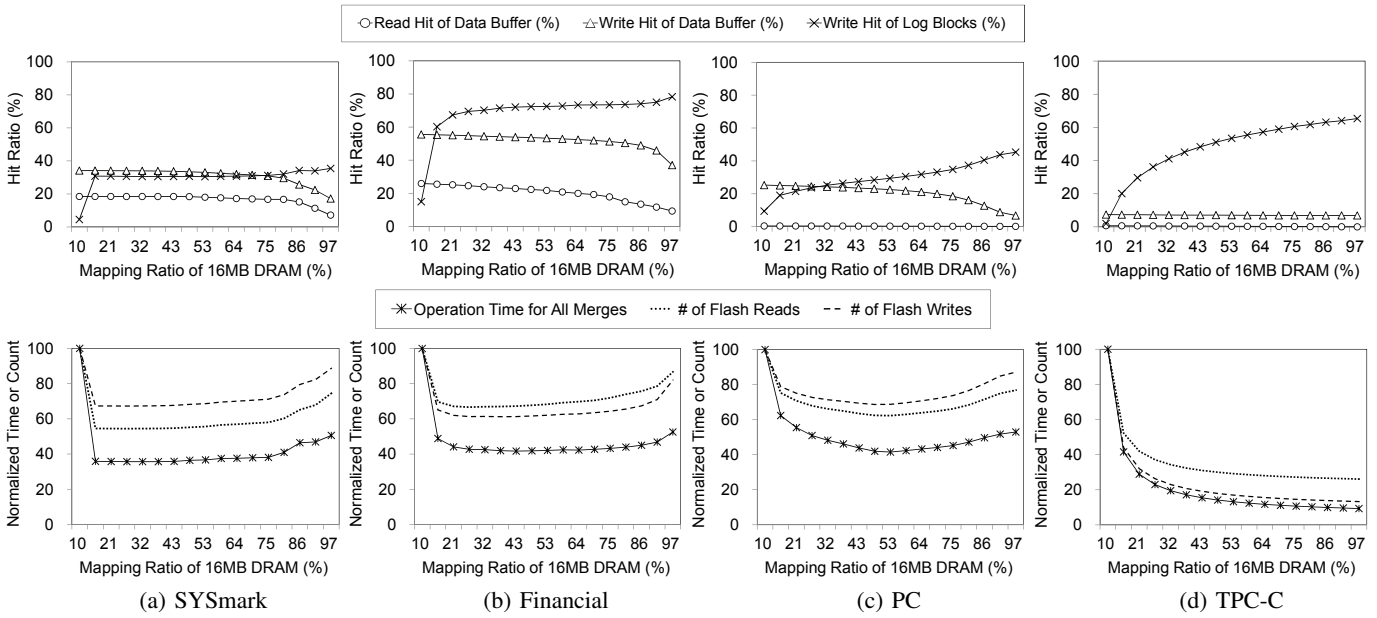


Fig. 7. Effects of the BM ratio with FAST

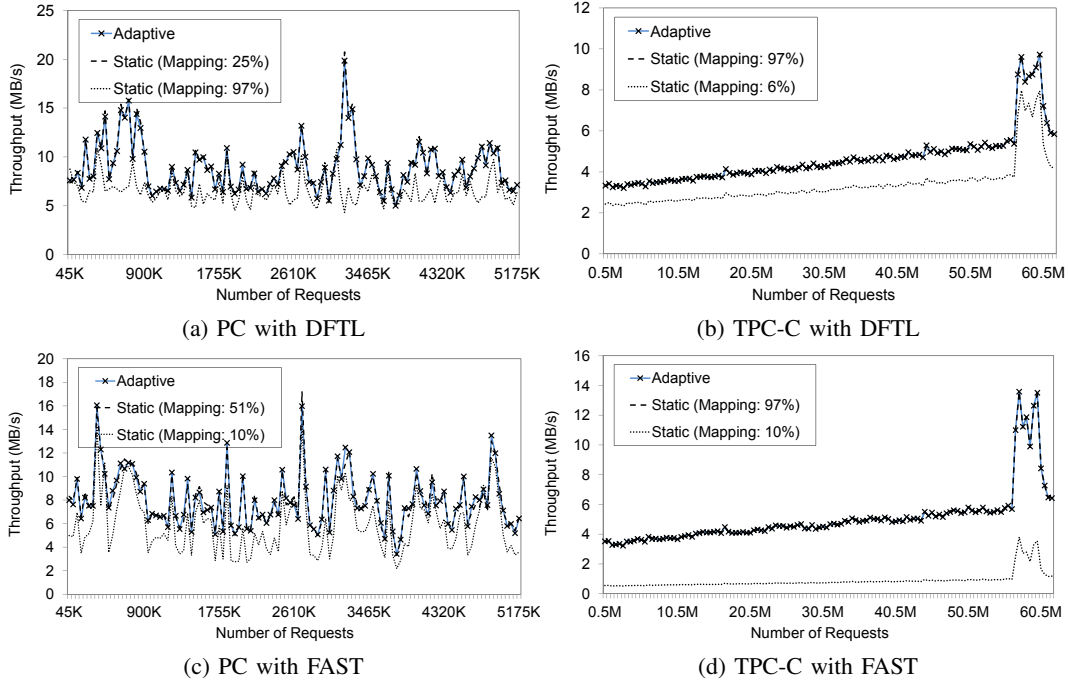


Fig. 8. Throughput variation comparison with DFTL and FAST

ratio of mapping as 50% of the DRAM size. For more specific analysis, we measured additional performance parameters with 16MB of DRAM under the static partitioning policy as shown in Fig. 7. First, the read and write hit ratios of the data buffer, and the write hit ratio of the log blocks are useful for observing trade-offs between buffering versus mapping. The number of write hits in the log blocks is counted when the valid pages in the log blocks are invalidated by following write requests. The effects of adjusting the BM ratio are revealed by additional parameters such as the number of flash reads and flash writes,

and the merge operation time, which are normalized to their largest values.

In the SYsmark and Financial traces as shown in Fig. 6(a), (b) and Fig. 7(a), (b), the best performance is obtained at the small mapping ratio due to their small working set size. As the mapping space increases from zero to around 25%, the write hit ratio in the log blocks is dramatically raised. As a result, valid page migration for merge operations is considerably prevented. In the meantime, the read and write hit ratios are nearly unchanged in spite of the diminished buffering space.

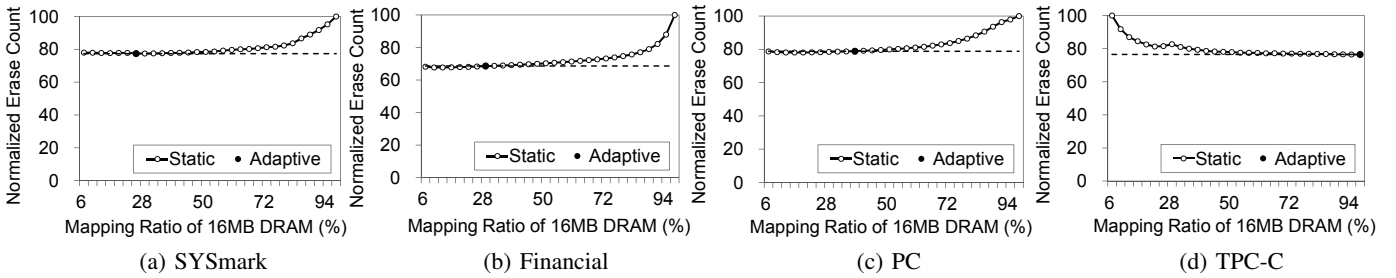


Fig. 9. Erase count comparison under various workloads with DFTL

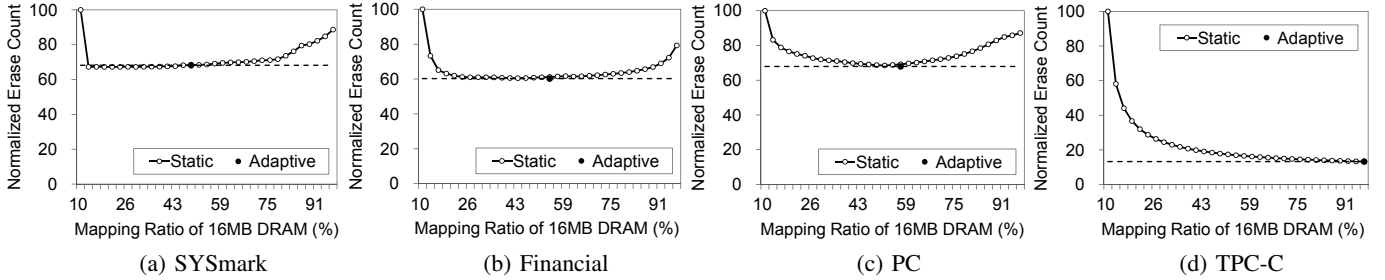


Fig. 10. Erase count comparison under various workloads with FAST

As the mapping space becomes greater than the working set size, however, additional log blocks bring no profit any more, and the reduced buffer hit ratios generate more flash operations including garbage collection overheads.

In our approach, when there are insufficient log blocks to accommodate the working set of writes, many valid page copies occur and are inserted into the ghost log blocks. The newly-inserted log pages make the cost-benefit of the ghost log blocks larger than that of the ghost buffer, thus extending the actual mapping space. When the log blocks are enough, valid page copies almost disappear, and the cost-benefit of the ghost log blocks is getting smaller. In this way, the proposed scheme tracks the most cost-beneficial point between buffering and mapping.

In the PC trace shown in Fig. 6(c) and Fig. 7(c) with the static partitioning policy, the operation time for merge operations is reduced as the number of log blocks increases because of the higher possibility of invalidating valid pages in log blocks. However, the profit of increasing log blocks is connected to decreasing the read and write hit ratio in the data buffer. The proposed scheme adjusts the BM ratio to the most cost-beneficial point that creates balance between the profit of reducing valid page copies and that of increasing buffer hits. In the TPC-C trace in Fig. 6(d) and Fig. 7(d), the buffer hit ratios are quite low and almost unchanged even with the very small buffering space, while the larger mapping space makes an opportunity for updating the pages in log blocks. According to the characteristics of this workload, the proposed scheme autonomously moves the BM ratio to the higher mapping ratio, and achieves the best performance. In addition, those results demonstrate that the adaptive partitioning scheme also works well with hybrid FTLs.

#### D. Throughput

Fig. 8 shows the throughput variance according to the periods of requests with 16MB of DRAM when using DFTL and FAST. For this simulation, we used the PC and TPC-C traces whose duration is long enough. We compared the throughput of the proposed scheme with those of the best and the worst cases of the static partitioning scheme. The throughput of the proposed scheme approximates the throughput of the best result of the static partitioning scheme for each period of requests. From this result, we believe that the proposed scheme can follow the optimal BM ratio in varied workloads without workload-specific static configurations or prior examination of the workload characteristics. This means that SSDs equipped with the proposed scheme can be widely adopted in different environments ranging from desktop to server storage systems running various user applications.

#### E. Erase Count

In NAND flash memory, the number of erase operations per block is limited, typically, from 5,000 to 100,000 [13], [14]. If a block is erased more than the limited number, the block is likely to be worn out and cannot be written any more due to frequent data errors. Considering this constraint, we need not only to distribute erase operations into all blocks, but also to reduce the number of erase operations. Fig. 9 and Fig. 10 present the number of erase operations with 16MB of DRAM under the DFTL and FAST schemes, respectively. The results of the adaptive partitioning scheme come near the lowest erase counts under the static partitioning policy.

In the DFTL and FAST schemes, the erase count is closely related to the number of write operations. In DFTL, the proposed scheme efficiently increases the write hit ratio in CMT, without sharply decreasing the write hit ratio of the data

buffer. Consequently, the proposed scheme reduces valid page copies for reclaiming data and translation blocks. In FAST, the proposed scheme helps FTL keep a proper number of log blocks for the working set size. As a result, FAST exploits the temporal locality of writes enough only with an acceptable decline of the write hit ratio in the data buffer. From those results, we indicate that the adaptive partitioning scheme can help extending the lifetime of SSDs.

## V. CONCLUSION

In the storage market, migration from hard disk drives to NAND flash-based storage devices is being accelerated. This migration is supported by many studies about internal software and hardware components: buffer management policies, Flash Translation Layer (FTL) algorithms, and multi-channel architectures to overcome the limitations of NAND flash memory. One of them, the device cache, in SSDs is an important component that has a significant impact on the performance. The device cache is used for both buffering data and caching the FTL mapping information. We observed that the most cost-beneficial ratio of the buffering and the mapping space changes according to workload characteristics. In existing studies, however, it is assumed that the ratio of them is fixed, and thus they cannot have a flexible use of the device cache. In this paper, we proposed an *adaptive partitioning scheme* to store more cost-beneficial data into the device cache for better performance of SSDs.

We built an evaluation model for calculating the cost-benefit value of increasing the buffering or mapping space. Based on this model, the adaptive partitioning scheme adaptively tunes the ratio of the buffering and the mapping space at every pre-defined interval. To evaluate the performance of the proposed idea, we applied the cost-benefit model to the two widely-used FTLs: the DFTL and FAST schemes. Using the proposed scheme, we successfully obtained the performance near the best performance under the static partitioning policy with varied workloads. Moreover, we expect that our approach can be also applied to various caching-based FTL schemes and buffer management algorithms.

## ACKNOWLEDGMENT

This work was supported by the IT R&D Program of MKE/KEIT. [2010-KI002090, Development of Technology Base for Trustworthy Computing]

## REFERENCES

- [1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy. Design tradeoffs for SSD performance. In *Proceedings of USENIX Annual Technical Conference (USENIX '08)*, pp. 57-70, Boston, MA, USA, Jun. 2008.
- [2] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. *ACM SIGMETRICS/Performance*, pp. 181-192, Seattle, WA, USA, Jun. 2009.
- [3] A. M. Caulfield, L. M. Grupp, and S. Swanson. Gordon: Using flash memory to build fast, power-efficient clusters for data-intensive applications. In *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '09)*, pp. 217-228, Washington, DC, USA, Mar. 2009.

- [4] Ron Weiss. Exadata smart flash cache and the sun oracle database machine - An oracle white paper. *Oracle*, 2009.
- [5] V. Prabhakaran, T. L. Rodeheffer, and L. Zhou. Transactional Flash. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, pp. 147-160, San Diego, CA, USA, Dec. 2008.
- [6] X. Ding, S. Jiang, and F. Chen. A device cache management scheme exploiting both temporal and spatial localities. *ACM Transactions on Storage*, vol. 3, no. 2, article 5, Jun. 2007.
- [7] 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. 52, no. 2, pp. 485-493, May 2006.
- [8] H. Kim and S. Ahn. A buffer management scheme for improving random writes in flash storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08)*, pp. 239-252, San Jose, CA, USA, Feb. 2008.
- [9] N. Megiddo and D. S. Modha. ARC: A self-tuning low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*, pp. 115-130, San Francisco, CA, USA, Mar. 2003.
- [10] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the 15th ACM Symp. on Operating System Principles (SOSP)*, pp. 79-95, Copper Mountain Resort, CO, USA, Dec. 1995.
- [11] A. Gupta, Y. Kim, B. Urgaonkar. DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proceeding of the 14th international conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 229-240, Washington, DC, USA, Mar. 2009.
- [12] 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, article 18, Jul. 2007.
- [13] 1G x 8 Bit / 2G x 8 Bit / 4G x 8 Bit NAND flash memory (K9WAG08U1M) data sheets. *Samsung Electronics*, Nov. 2005.
- [14] 2G x 8 Bit NAND flash memory (K9GAG08UXM) data sheets. *Samsung Electronics*, Dec. 2006.
- [15] M.-L. Chiang, P. C. H. Lee, and R.-C. Chang. Using data clustering to improve cleaning performance for flash memory. *Software: Practice and Experience*, vol. 29, issue 3, pp. 267-290, Mar. 1999.
- [16] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho. A space-efficient flash translation layer for CompactFlash systems. *IEEE Transactions on Consumer Electronics*, vol. 48, no. 2, pp. 366-375, May 2002.
- [17] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee. A superblock-based flash translation layer for NAND flash memory. In *Proceedings of the 6th ACM International Conference on Embedded Software (EMSOFT '06)*, pp. 161-170, Seoul, Republic of Korea, Oct. 2006.
- [18] Y.-G. Lee, D. Jung, D. Kang, and J.-S. Kim.  $\mu$ -FTL: A memory-efficient flash translation layer supporting multiple mapping granularities. In *Proceedings of the 8th ACM International Conference on Embedded Software (EMSOFT '08)*, pp. 21-30, Atlanta, GA, USA, Oct. 2008.
- [19] S. Kang, S. Park, H. Jung, H. Shim, and J. Cha. Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices. *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 744-758, Jun. 2009.
- [20] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable DRAM alternative. In *Proceedings of the 36th International Symposium on Computer Architecture (ISCA)*, pp. 1-12, Austin, TX, USA, Jun. 2009.
- [21] J. H. Yoon, E. H. Nam, Y. J. Seong, H. Kim, B. S. Kim, S. L. Min, and Y. Cho. Chameleon: A high performance flash/FRAM hybrid solid state disk architecture. pp. 17-20, *IEEE Computer Architecture Letters*, vol. 7, no. 1, Jan.-Jun. 2008.
- [22] K. Sun, S. Baek, J. Choi, D. Lee, S. H. Noh, and S. L. Min. LFTL: Lightweight time-shift flash translation layer for flash memory based embedded storage. In *Proceedings of the 8th ACM International Conference on Embedded Software (EMSOFT '08)*, pp. 51-58, Atlanta, Georgia, USA, Oct. 2008.
- [23] Mark Russinovich. DiskMon for Windows v2.01, <http://technet.microsoft.com/en-us/sysinternals/bb896646.aspx>, Nov. 2006.
- [24] blktrace User Guide, <http://blogninja.com/doc/blktrace/blktrace.pdf>, Feb. 2007.