Clustered Page-Level Mapping for Flash Memory-Based Storage Devices

Hyukjoong Kim and Dongkun Shin, *Member*, IEEE

Abstract — *Recent consumer devices such as smartphones, smart TVs and tablet PCs adopt NAND flash memory as storage device due to its advantages of small size, reliability, low power consumption, and high performance. The unique characteristics of NAND flash memory require an additional software layer, called flash translation layer (FTL), between traditional file systems and flash memory. In order to reduce the garbage collection cost, FTLs generally try to separate hot and cold data. Previous hot and cold separation techniques monitor the storage access patterns within storage device, or exploit file system hints from host system. This paper proposes a novel clustered page-level mapping, called CPM, which can separate hot and cold data efficiently by allocating different flash memory block groups to different logical address regions. CPM can reduce the FTL map loading overhead during garbage collection and it does not require any high-cost monitoring overhead or host hint. This paper also proposes a K-associative version of CPM, called K-CPM, which allows different logical address regions to share a physical block group in order to achieve high block utilizations. Experimental results show that CPM improves the storage I/O performance by about 54% compared with a previous page-level mapping FTL, and K-CPM further improves the performance by about 19.4% compared with CPM*[1].

Index Terms — NAND Flash Memory, Flash Translation Layer, Clustered Page Mapping, Embedded Storage.

I. Introduction

NAND flash memory is widely used by mobile consumer devices such as tablet PCs and smartphones due to its several advantages: high speed, robustness, energy efficiency, and compact size. However, NAND flash memory is incompatible with traditional block devices due to its unique characteristics.

A NAND flash memory chip is composed of several blocks, and a block is composed of a bundle of pages. Whereas a block is the unit of an erase operation, a page is the unit of write or read operation. A page cannot be overwritten before the corresponding block is erased. The erase operation has a higher cost than the write and read operations. Each physical block has a limited number of program/erase cycles. The pages in a block should be programmed sequentially [1].

[1] This work was supported by the ICT R&D program of MSIP/IITP. [10041244, SmartTV 2.0 Software Platform].

Hyukjoong Kim and Dongkun Shin are with the College of Information & Communication Engineering, Sungkyunkwan University, Suwon, Korea (e-mail: wangmir, dongkun@skku.edu).

In order to provide a traditional block device interface by hiding these unique characteristics, a software layer, called flash translation layer (FTL), is used between traditional file systems and flash memory. FTL translates a logical address of file system into a physical address of flash memory chip. For the address translation, FTL manages an address mapping table. When a write request arrives, FTL writes the data at a clean page, and updates the mapping table. If another physical page has an old data for the target logical address, the page is invalidated. When there are little free pages, FTL invokes the garbage collection (GC) in order to reclaim the invalid pages.

The address mapping schemes of FTLs can be classified into the block-level mapping and the page-level mapping [2]. The block-level mapping [3] first calculates the logical block number (LBN) and the page offset from a given logical page number (LPN). The mapping table manages the translation between an LBN and the allocated physical block number (PBN). An LBN can be mapped into any PBN. However, a logical page should be written at the same page offset within the mapped physical block, which is called the *in-place write scheme*. If an update request is sent, the block-level mapping should allocate a new clean block, and should copy all the valid pages in the original block into the corresponding page offsets of the new block. Therefore, its write performance is significantly low.

In the page-level mapping [4], a logical page can be written at any page offset within a physical block, which is called the *out-of-place write scheme*. Therefore, the page-level mapping should manage the address translation between an LPN and the mapped physical page number (PPN). Compared with the block-level mapping, the page-level mapping shows better performance but requires a larger size of mapping table.

The hybrid mapping techniques [5]-[7] are intermediate schemes. The hybrid mapping allocates a data block for each logical block. In addition, several physical blocks are allocated as log blocks, where incoming data are first written. Whereas normal data blocks are managed by the in-place scheme, the log blocks are managed by the out-of-place scheme. Therefore, the write performance of hybrid mapping is similar to the performance of page-level mapping. Since the number of log blocks is small, the size of mapping table is similar to that of block-level mapping. When a new log block should be allocated, the hybrid mapping should copy all valid pages in victim log blocks into the associated data blocks. This operation is called log block merging. The block merge cost is significantly high especially when a workload is random-write dominant.

Contributed Paper
Manuscript received 01/01/15
Current version published 03/30/15
Electronic version published 03/30/15.
0098 3063/15/$20.00 © 2015 IEEE

In the page-level mapping scheme, if hot and cold data are written into different physical blocks, the GC cost can be reduced. Since the hot data will be frequently updated, the physical block allocated for hot data will have many invalid pages and it can be a low-cost victim block for GC. However, unless any hot/cold separation technique is used, hot and cold data can be mixed within a physical block since the incoming data will be written in the order of request arrival time. Then, the GC cost will be high since many cold pages should be copied during GC. Several previous studies have proposed the hot/cold data separation techniques [8]-[13]. However, these works should monitor the data access pattern to identify hot data, or require an extended storage interface to transfer semantic hints on the written data.

Another possible technique for hot/cold separation is to exploit the logical addresses of data. Generally, file systems tend to allocate different logical address regions to different file types or directories. The blocks of the same files or directories are allocated at adjacent logical addresses [14]. Therefore, if data are written at different physical blocks based on the logical addresses of the data, the hot and cold data will be stored at different physical blocks. The block mapping and hybrid mapping schemes allocate different physical blocks based on the logical address. However, their in-place write schemes make too high GC costs.

This paper proposes a clustered page-level mapping, called CPM, which allocates different flash memory block groups to different logical address regions. CPM divides the entire logical address space into several clusters of the same size. Each cluster is assigned with a fixed number of physical blocks, and only the logical pages included at the address range of the cluster can be written at the allocated physical blocks. The physical blocks are managed by the out-of-place scheme. By allocating different physical blocks based on the logical addresses, CPM can separate hot and cold data implicitly, and thus it can reduce the map loading overhead and page copy cost during the GC.

However, CPM can suffer from the low utilization of physical blocks. Even when other clusters have free pages within their allocated physical blocks, the GC should be invoked if the target cluster has no free pages. In particular, the clusters allocated for cold data will hold many free pages that are not used for a long period.

To solve the problem of CPM, this paper also proposes K-associative clustered page-level mapping, called K-CPM, which allows a cluster to share its physical blocks with other clusters. Since the physical block sharing can increase the GC cost, the number of clusters which can share physical blocks with a cluster is limited in K-CPM. Therefore, K-CPM can improve the utilization of physical blocks without increasing the GC cost significantly.

Experiments with a flash memory simulator show that CPM reduces the GC cost by about 54% compared with the previous page-level mapping. K-CPM reduces the GC cost further by about 19.7% compared with CPM by increasing the block utilization. Moreover, K-CPM reduces the write latencies by delaying GCs.

II. Related Works

Several hybrid mapping FTLs have been proposed. BAST [5] uses 1-to-1 mapping between data block and log block. It is simple but suffers from low utilizations of log blocks, called the log block thrashing problem. FAST [6] solved the problem by sharing a log block among multiple data blocks, which is called 1-to-N mapping. However, FAST has a high log block merge cost especially when a victim log block has many associated data blocks. KAST [7] allows only a limited number of data blocks to be associated with a log block in order to limit the log block merge cost.

Superblock FTL [15] is also a hybrid mapping scheme. It groups several contiguous logical blocks into a *superblock*, and allocates several physical blocks as data blocks. Several log blocks can be allocated for each superblock to handle write requests on the superblock. In Superblock FTL, data blocks as well as log blocks are managed by the out-of-place scheme. Therefore, it can alleviate the log block merge overhead. In addition, Superblock FTL utilizes the temporal information at log block merge operations in order to gather cold data into data blocks. The least-recently-used (LRU) log block is selected for a victim rather than the log block with the maximum number of invalid pages.

However, Superblock FTL cannot completely overcome the inherent limitation of hybrid mapping, i.e., data blocks and log blocks should be separated. An incoming data should always be written at a log block first, and the data blocks are used for only the valid page copy operations during the GC. Therefore, the utilizations of both data blocks and log blocks are low. In the proposed CPM technique, data blocks and log blocks are not differentiated. Therefore, the block utilization can be improved at CPM compared with Superblock FTL. A comparison between these two schemes will be discussed later in detail.

DFTL [16] is a page-level mapping for resource-constrained devices. DFTL uses the on-demand caching technique for the page-level mapping table. Whereas the entire page-level mapping table is stored in the NAND flash memory, only the recently referenced map entries are loaded into RAM. For a map entry miss, multiple logically-contiguous map entries are loaded into RAM. Therefore, DFTL suffers from a high map loading overhead especially when the data access pattern has little temporal and spatial localities. In addition, if the logical addresses of valid pages in the victim block of GC have low spatial localities, the GC cost increases due to the map loading overhead.

There are several hot/cold separation techniques for page-level mapping FTLs. Park *et al*. proposed a hot/cold separation scheme which considers not only the frequency but also the recency of data. The scheme should manage an additional table for gathering the frequency and recency of each data [9]. Jung *et al*. proposed a process-aware hot/cold separation technique where the data access pattern is determined by the process that sends the write requests [10]. CAT [11] considers the age of a physical block at the victim block selection during GC. By avoiding selecting a young block, it can wait for the additional invalidations of hot data in the young block.

LAST [8] is a hot/cold separation technique for the log blocks in the hybrid mapping. It utilizes the data update intervals to identify hot data. ComboFTL [12] uses the hot/cold separation technique to determine the data location at hybrid storage device, which has a multi-level cell (MLC) region and a single-level cell (SLC) region. ComboFTL determines the hotness of data based on the request size. The data with small I/O sizes are considered as hot data, and they are stored at the SLC region. Wu *et al*. [13] proposed a file system-aware hot data separation technique. Since the file system metadata are frequently updated by file system operations, the technique handles the file system metadata as hot data.

Most of the previous works require additional profiling cost, or require new interfaces between host and storage in order to transfer data semantics. However, the proposed CPM can efficiently separate hot and cold data without any profiling cost or new storage interface.

III. MOTIVATION

A. Locality Analysis on Real Workloads

To design a logical address-based hot/cold separation technique, the real storage I/O workloads are observed. Fig. 1 shows the write count for each logical page in a real workload, which is collected from a Linux-based smartphone. From the graph, the frequently accessed regions and infrequently accessed regions are explicitly identified. Whereas the address regions in the area of (a) have significantly high update counts, the address regions in the area of (b) have the update counts less than 10. That is, different address regions have different update frequencies. Therefore, it can be known that a hot and cold separation technique based on logical address can be effective.

The correlation between the hotness and the logical address of data results from the block allocation scheme of file systems. For example, the ext4 file system [14] separates the metadata from user data in the logical block address. In addition, ext4 tries to allocate adjacent blocks for a file or multiple files under a directory to reduce the hard disk seek time. Since current smart consumer devices tend to store the same type of data into the same directory, it is likely that the data in adjacent logical addresses have similar write patterns.

For a detailed analysis, the address access pattern of an application is observed as shown in Fig. 2. The graph shows the written logical addresses during the execution of a social network service (SNS) application. Whereas the address range in (a) has the average update count of 1.2, the average update count of the range (b) is 30.4. Therefore, the address region of (a) is cold and the address region of (b) is hot. The address ranges of (a) and (b) correspond to the cached image folder and the database folder, respectively. The database files are used for managing image file information. The image files have read-intensive data, but the database files are frequently updated during the read or write operations on the image files. Therefore, the storage device will receive interleaved write requests on hot region and cold region.

Such interleaved updates on hot and cold regions are significantly harmful to the page-level mapping. Since the page-level mapping FTL allocates the physical block in the order of request arrival time, the pages sent in a similar time will be adjacent in the physical address space. Therefore, hot and cold data are mixed within a physical block and thus the GC cost increases. If the physical block of data is allocated considering the logical address of the data, it is more probable that the hot and cold data will be located at different physical blocks.

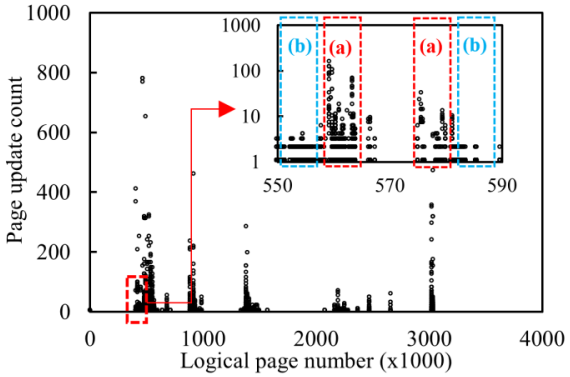

Fig. 1. Write counts on different logical pages (smartphone trace).

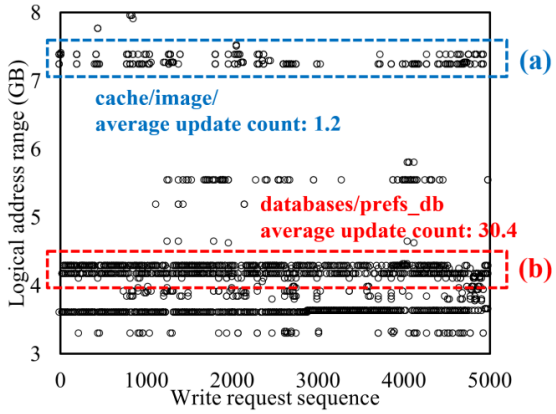

Fig. 2. Write pattern during SNS workload.

B. Demand Map Loading in DFTL

If logically separated pages are mixed within a physical block in DFTL, the map loading cost as well as the page copy cost during the GC increases. In the page-level mapping, the mapping entry for a logical page includes the PBN and the page index (PI) of the mapped physical page. The size of a mapping entry for a logical page can be calculated as follows:

$$log_2 N_B^S + log_2 N_P^B \ _{byte} = \ log_2 N_P^S \ _{byte} \qquad (1)$$

N_B^S and N_P^S are the number of total physical blocks and the number of total physical pages in the storage device, respectively. N_P^B is the number physical pages per block.

If the page size is 8 KB, the size of a mapping entry is 4 bytes for the storage whose capacity is less than 32 TB. Since read and write operations are performed in the unit of page in flash memory, multiple logically-contiguous map entries in a flash memory page are loaded into SRAM at a map entry miss in order to utilize the spatial locality on data access pattern. The page with multiple map entries is called virtual translation page

(VTP). The VTPs are stored in the map blocks which are separated from normal data blocks. The global translation directory (GTD) is the mapping table for finding the physical locations of VTPs. Each entry in GTD has the PPN for a virtual translation page number (VPN). Whereas the normal page map entries are selectively loaded into the cached map table (CMT) of SRAM, GTD resides on a fixed map table (FMT) of SRAM.

Fig. 3 shows the architecture of DFTL. It is assumed that one block is composed of four pages, and a single VTP has the mapping entries for four logically-contiguous pages. CMT can cache two VTPs. The block valid page count (BVC) table manages the numbers of valid pages in data blocks in order to select a GC victim block based on the number of valid pages at data blocks. B_i denotes the physical block whose PBN is i, and P_j denotes the logical page whose LPN is j. V_k is the VTP whose VPN is k.

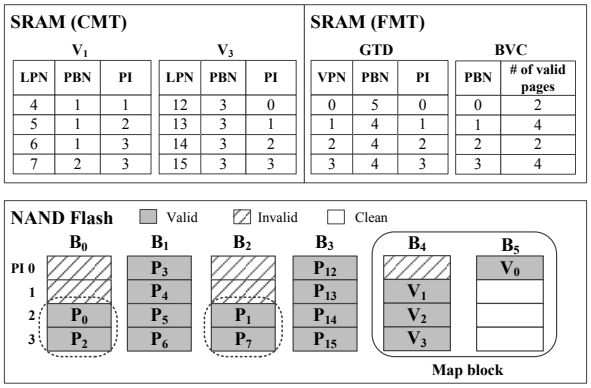

Fig. 3. Mapping table management in DFTL.

In Fig. 3, the GC should be started since no data block has any free pages. The physical block with the maximum number of invalid pages is chosen as a victim block for the GC. Therefore, B_0 and B_2 are candidates. Whereas all the map entries of valid pages in B_0 can be found at one VTP, V_0, the map entries of valid pages in B_2 are scattered at two VTPs, V_0 and V_2. During GC, the map entries of valid pages should be modified since the physical pages of the valid data are changed. Therefore, more VTPs should be loaded into CMT during GC if B_2 is selected for a GC victim.

From this example, it can be known that a low spatial locality among the valid pages in the GC victim block can increase the GC overhead due to the frequent map loading operations. However, the pages in a physical block may not be spatially adjacent in DFTL.

IV. CPM: CLUSTERED PAGE-LEVEL MAPPING

A. Architecture

This paper proposes a clustered page-level mapping (CPM) scheme in order to increase the spatial adjacency of the pages within a physical block. It can separate hot and cold data into different physical blocks, and can reduce the map loading overhead during GC. CPM divides the overall logical address space into multiple clusters, each of which is composed of S_C number of logically contiguous blocks. CPM allocates several physical blocks to each cluster, and the physical blocks are managed by the out-of-place scheme. The physical blocks allocated for a cluster can have the logical pages belong to the cluster. Therefore, each physical block has logically adjacent pages.

CPM also uses the demand map loading technique like DFTL. Since the logical address space of the pages in a physical block is limited, the number of VTPs to be updated during GC is also limited. Therefore, the map loading overhead during GC can be reduced.

CPM is similar to Superblock FTL since both of them divide the logical space into multiple regions and use the out-of-place scheme in all physical blocks. However, CPM does not distinguish between data blocks and log blocks, therefore, it can enhance the block utilization and can reduce the GC cost compared to Superblock FTL.

Fig. 4 compares between the GC operations in Superblock FTL and CPM. It is assumed that the size of superblock or cluster is two logical blocks, and each superblock or cluster can use up to 4 physical blocks. Whereas two data blocks, D_0 and D_1, and two log blocks, U_0 and U_1, are allocated in the Superblock FTL, four data blocks, B_0 to B_3, are allocated in the CPM FTL. The physical blocks of the two schemes are equally filled with valid or invalid pages, and both the schemes require the GCs since there are no free pages to write incoming data. Since Superblock FTL cannot write incoming data at data block, the GC should generate a free block to be used for an update block. Therefore, the GC should select two victim blocks to be erased, one of which will be used for an update block and another will be used to copy the valid pages in victim blocks. In this example, Superblock FTL selects D_0 and D_1 as victim blocks, copies all the valid pages into a new data block D_0, and allocates a new log block U_2. However, CPM selects only one victim block, B_0, and copies one valid page into the newly allocated block, B_4. The block B_4 can be used for incoming data. Therefore, CPM can utilize the physical blocks more efficiently, and can reduce the GC overhead compared to Superblock FTL.

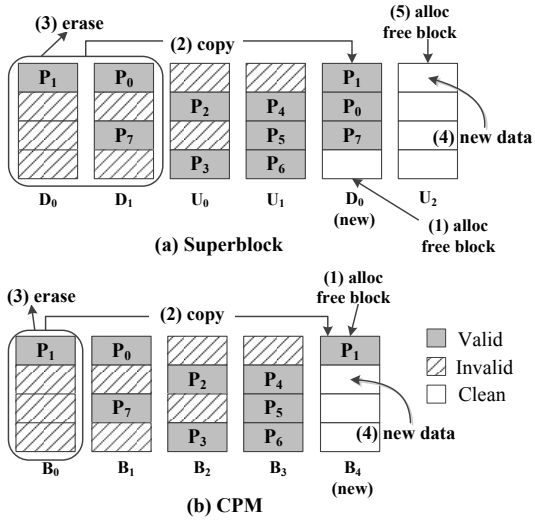

Fig. 4. Comparison between GCs of Superblock FTL and CPM.

CPM allows more number of physical blocks to be allocated than the logical size of a cluster. In this paper, C_i denotes the cluster whose cluster number is i, and $N_B(C_i)$ denotes the number of physical blocks allocated for C_i. Since a larger value of $N_B(C_i)$ requires more entries for PBNs, the maximum number of $N_B(C_i)$ is limited to $N_{B\,max}^{C}$ in order to limit the size of the mapping table.

Fig. 5 describes the structure of CPM. It is assumed that $S_C = 2, N_B\ C_0\ = 3$, and $N_B(C_2) = 2$. CPM manages two mapping tables: one is the logical-to-physical page mapping table (PMT) and another is the cluster-block mapping table (CBMT). There are separated map blocks in the flash memory for these mapping tables like DFTL. The PMT manages the PPN (PBN and its PI) for each LPN. CBMT has the information on physical blocks allocated for each cluster. There are $N_{B\,max}^{C}$ number of entries for each cluster in CBMT, where each entry has the PBN and valid page count for an allocated physical block.

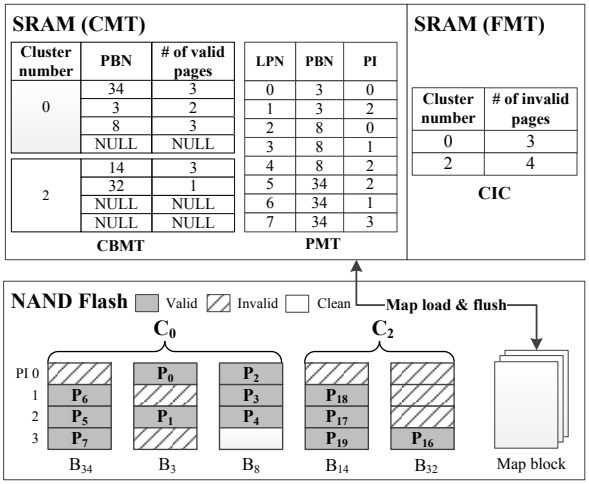

Fig. 5. Mapping table and data management structure of CPM.

PMT and CBMT are loaded into CMT on demand, and each table is managed by the LRU replacement policy. The mapping entries in PMT are loaded into CMT in the unit of VTP, i.e., multiple logically contiguous mapping entries are loaded at a time. However, each cluster-block mapping entry is the unit for replacement and loading in CBMT since one entry covers a large logical address space (e.g., 16 MB).

DFTL should maintain the valid page counts for all the physical blocks in SRAM in order to select a GC victim block. However, CPM only needs to maintain the cluster-level invalid page count (CIC) for each cluster since CPM first selects a victim cluster, and then selects victim blocks within the victim cluster using the valid page counts in CBMT.

B. Read & Write Operations

If host sends a read request, CPM searches the mapping entry of the logical page in PMT. For example, P_0 is stored in B_3 and the page index is 0 in Fig. 5. If host sends a write request, CPM should find a free page for the new data from the physical blocks allocated for the cluster, and then should update the corresponding mapping entry in PMT and the valid page counts in CBMT. However, if there is no free page in the allocated blocks, a new free block should be allocated for the cluster. If there are no available free block, or $N_B\ C_i \leq N_{B\,max}^{C}$, CPM invokes the GC to make free blocks.

CPM can use two different GC techniques. The first one is the intra-cluster garbage collection (IntraGC), which reclaims invalid pages within a cluster that needs free pages. The IntraGC operation is similar to the GC of normal page-level mapping. Fig. 6(a) describes the procedure of IntraGC. When a GC is invoked by the cluster of C_0, the CPM chooses B_0 as a victim block since it has the maximum number of invalid pages. The number of victim blocks is always only one. CPM copies valid pages into the free block B_8, which is reserved for GC operation. B_8 becomes a new physical block allocated for C_0, and the entry of C_0 in CBMT is updated. The erased block B_0 is reserved for future GCs.

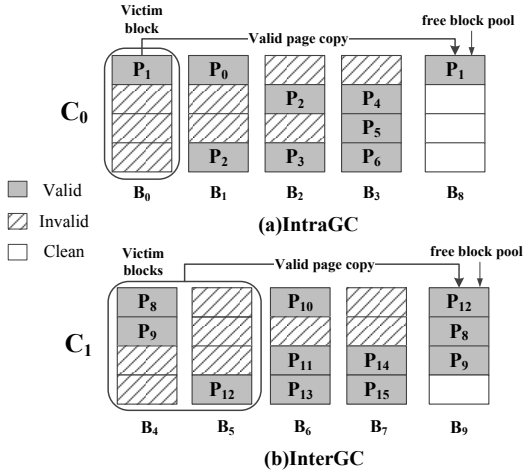

Fig. 6. IntraGC and InterGC operations in CPM.

The second GC technique is the inter-cluster garbage collection (InterGC), which reclaims invalid pages in an external cluster (called victim cluster) rather than the target cluster. Since CPM separates the logical address space into clusters, the physical block allocated for a cluster cannot be used for other clusters. Therefore, InterGC should make a whole free block from a victim cluster in order to give the free block to the target cluster. Fig. 6(b) shows the InterGC operation. First, InterGC finds a victim cluster that has the maximum number of invalid pages from the CIC table. Then, InterGC selects one or more victim blocks from the victim cluster by scanning the fields of valid pages in CBMT. Multiple victim blocks can be selected in order to make a whole free block. In Fig. 6(b), B_4 and B_5 are selected. CPM copies the valid pages (P_{12}, P_8, and P_9) into the free block (B_9) that is reserved for GC operation. Two victim blocks, B_4 and B_5, can be erased. One is allocated for the target cluster, and the other is reserved for future GCs. Compared with IntraGC, InterGC has a higher GC cost since it should generate a whole free block.

If $N_B\ C_i\ = N_{B\,max}^{C}$, C_i cannot use more physical blocks. Therefore, IntraGC should be used. Otherwise, CPM chooses IntraGC or InterGC considering the GC efficiency, ε_{GC}, which is the number of generated free pages per GC cost, and can be

represented as follows:

$$\varepsilon_{GC} = \frac{N_{P_{free}}}{N_{P_{valid}} \times (t_{prog} + t_{read}) + N_{B_{victim}} \times t_{erase}} \quad (2)$$

t_{prog}, t_{read} and t_{erase} denote the latencies for program, read, and erase operations, respectively. $N_{P_{free}}$ is the number of free pages generated by the GC. $N_{P_{valid}}$ is the number valid pages to be copied during the GC. $N_{B_{victim}}$ represents the number of victim blocks during the GC. CPM compares the GC efficiencies of IntraGC and InterGC, and selects a more efficient GC.

C. Problems of CPM

Although CPM can efficiently reduce the GC cost by allocating separated physical blocks to different clusters, the policy will decrease the block utilization. Therefore, the GC for a cluster can be invoked even though there are free pages in the physical blocks allocated for other clusters. Such an early GC will copy the pages that will be invalidated in the near future.

In order to mitigate the low block utilization problem, a large size of clusters will be beneficial since many logical blocks can share a physical block. However, if the size of cluster is too large, the hot and cold will be mixed within a cluster. In addition, the number of VTPs required for storing all the mapping entries for a cluster will increase, and thus the map loading overhead during GC also will increase. Therefore, another optimization technique is required, which can improve the block utilization without increasing the cluster size.

V. K-CPM: K-Associative Clustered Page-level Mapping

A. Architecture

The K-associative clustered page-level mapping (K-CPM) allows several clusters to share a physical block. When a cluster needs free pages, K-CPM does not perform the GC immediately, but it checks whether there is any other cluster that can share its physical blocks with the target cluster. By allowing physical block sharing, K-CPM can improve the block utilization and can delay the GC. However, the map loading overhead during the GC on the shared physical block may increase. Considering this problem, K-CPM limits the maximum number of clusters which share the physical blocks allocated for a cluster.

When a logical page P_j belongs to a cluster C_i, C_i is called the *owner cluster* of the logical page, and it is represented as $C_{own}\ P_j = C_i$. When a logical page P_j is written at the physical block allocated for the cluster C_k, C_k is called the *saved cluster* of the logical page, and it is represented as $C_{saved}\ P_j = C_k$. If the owner cluster and the saved cluster of a logical page are different, it is called an *adopted page*. If $C_{own}\ P_j = C_i$ and $C_{saved}\ P_j = C_k$ for a certain P_j, C_i is called an *associated cluster* (AC) of C_k, and the number of ACs of C_k is represented as $N_\alpha(C_k)$. In the case, C_k is called a *distributed cluster* (DC) of C_i, and the number of DCs of C_i is represented as $N_\delta(C_i)$.

K-CPM has a restriction on the number of associated clusters such that $N_\alpha\ C_i \leq K$ for all clusters. As a larger value of K is used, the block utilization is improved. However, the map loading overhead during GC increases, and the number of bits required for representing physical block index increases. By limiting the maximum number of associated clusters, K-CPM can manage the map loading overhead and the mapping table size. However, there is no limit on the number of distributed clusters of a cluster since it does not directly affect the GC overhead.

Fig. 7 shows the overall architecture of K-CPM. The basic assumption of the figure is the same as Fig. 5. The physical blocks allocated for C_0 and C_2 have adopted pages. For example, whereas the owner cluster of P_{10} is C_1, the saved cluster of P_{10} is C_0. The associated clusters of C_0 are C_0, C_1, and C_2 (i.e., $N_\alpha\ C_0 = 3$). P_2 and P_3, whose owner cluster is C_0, are saved in the physical blocks of C_2. Therefore, C_2 is a distributed cluster of C_0 and $N_\delta\ C_0 = 2$.

In order to manage the associated and distributed clusters, K-CPM has the cluster association map table (CAMT) and the cluster distribution map table (CDMT), all entries of which reside in the FMT of SRAM. The usages of CBMT and PMT are the same as CPM. However, each mapping entry in PMT has the cluster number of PBN additionally. The CAMT manages the associated cluster number (ACN) for each cluster, and it also manages the number of adopted pages of each AC. The CDMT manages the distributed clusters for each cluster. The distributed clusters of each cluster are managed with a linked list since the number of distributed clusters of a cluster is variable.

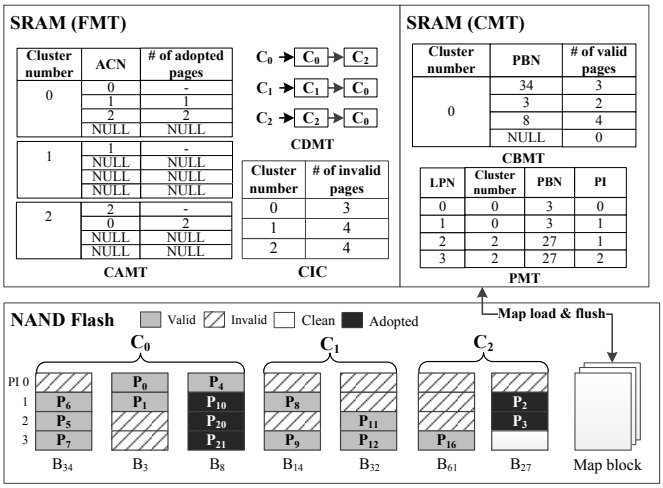

Fig. 7. Mapping table and data management structure of K-CPM.

B. Write Operation

Whereas the read operation of K-CPM is the same as CPM, the write operation of K-CPM is quite different with that of CPM. There are four different cases which should be handled differently in K-CPM for a write request on the target cluster C_{req}.

Case 1: If there are free pages in the physical blocks allocated for C_{req}, the write request can be handled within C_{req}. K-CPM updates the corresponding mapping entry in PMT and the number valid pages in CBMT.

Case 2: If C_{req} has no free pages, but a free physical block can be allocated for C_{req} (i.e., $N_B(C_{req}) < N^C_{B_{max}}$), then K-CPM allocates a free block for C_{req} and the incoming data are written at the block.

Case 3: If C_{req} has no free page and a new physical block cannot be allocated (i.e., $N_B(C_{req}) = N^C_{B_{max}}$), K-CPM checks whether one of DCs of C_{req} has free pages. If a DC has free pages, then the incoming data are sent to the DC. No new entry is inserted into CAMT or CDMT. Only the field of adopted page count in CAMT is updated.

Case 4: If there are no free pages in the DCs of C_{req}, K-CPM should find a new distributed cluster for C_{req}. If C_i has free pages and $N_\alpha\ C_i$ is less than K, C_i can be a new DC of C_{req}. Then, C_{req} is inserted as an AC of C_i in CAMT, and C_i is inserted as a DC of C_{req} in CDMT.

If K-CPM fails to find a new distributed cluster, the GC should be performed to reclaim invalid pages.

C. Garbage Collection

K-CPM also selects IntraGC or InterGC considering the GC efficiency. However, the physical blocks allocated for the DCs of the target cluster can be a victim block in the IntraGC of K-CPM since K-CPM can send the write request to the DCs of the target cluster.

InterGC selects a victim cluster from all clusters but the DCs of the target cluster, and it generates a free block to be allocated for the target cluster or its DCs. If all of the target cluster and its DCs have been allocated with $N^C_{B_{max}}$ number of physical blocks respectively, the free block generated by InterGC cannot be allocated for the target cluster or its DCs. In this case, K-CPM should assign a new DC that has less than $N^C_{B_{max}}$ number of allocated physical blocks and its associativity is less than K.

As an extreme case, K-CPM cannot find a new DC if the numbers of ACs of all clusters are K. To handle such a case, K-CPM uses K-InterGC, which reduces the associativity of a cluster by removing its associated clusters. K-InterGC finds a cluster which has the minimum number of adopted pages at one of its DCs, and moves the adopted pages to the cluster or its other DCs. Then, K-InterGC can obtain a cluster whose associativity is less than K. During the page migration, other GCs can be invoked. Therefore, K-InterGC has a significantly high GC cost. However, since K-InterGC hardly occurs, the high cost of K-InterGC may not be a significant problem.

VI. Experiments

To demonstrate the effectiveness of the proposed FTL schemes, three different page-level mapping schemes, DFTL, CPM and K-CPM, are compared with an FTL simulator.

A. Comparison on Mapping Table Size

TABLE I compares the memory sizes for mapping tables under DFTL, CPM and K-CPM. It is assumed that the total storage capacity is 32 GB, the page size is 8 KB, the number of pages per block is 128, the cluster size (S_C) in CPM and K-CPM is 16 MB, and $N^C_{B\ max}$= 32. CPM requires 192 KB of additional memory space for CBMT compared to DFTL. K-CPM requires 256 KB of additional space for CBMT, CAMT and CDMT. As the amount of increased space for mapping tables, the competition on the limited space of SRAM will be severe and the map loading overhead will be increased. However, the increased memory space can be negligible compared to the size of PMT.

TABLE I
Mapping Table for each Scheme (32GB Storage)

	PMT	CBMT	CAMT	CDMT
DFTL	16MB			
CPM	16MB	192KB		
K-CPM	16MB	192KB	32KB	32KB

B. Workloads

For experiments, real workloads are used, which are collected at a smartphone while executing several applications. The smartphone uses a Linux-based mobile platform, and the file system is ext4. The blktrace is used to collect storage I/O traces. TABLE II shows the characteristics of seven workloads. The *AppInstall* trace is collected while installing top twenty ranking applications, and the *AppUpdate* trace is collected while updating the installed applications. The *AppLaunch* trace is collected while executing the installed applications, and the *Browser* trace is extracted during one hour of web surfing. The *SNS* trace is also extracted during one hour of browsing SNS pages, and the *Map* trace has 30 minutes of map searching operations. The *Mp3copy* trace is collected while copying forty MP3 files from a PC to the smartphone via USB interface.

TABLE II
Working Set (WS) Analysis For Real Workloads

	No. of WS	Avg. WS size (MB)	Avg. IO size (KB)	Avg. page update	Write ratio (%)
AppInstall	58	10.93	84.0	1.20	79.2
AppUpdate	34	8.15	31.8	1.90	18.5
AppLaunch	59	12.34	66.6	1.25	53.2
Browser	33	6.81	27.4	3.35	84.1
SNS	33	3.87	9.97	4.81	31.7
Map	14	8.56	10.2	2.67	84.5
Mp3copy	68	15.74	295.4	1.06	96.8

The accessed address regions in each trace are divided into several working sets. The working set is a logically contiguous address region that is composed of pages with similar access patterns. In order to remove too small working sets, two small working sets are merged into a working set, if the distance between them is less than 512 KB.

As shown in the table, *AppInstall, AppLaunch*, and *Mp3copy* are composed of many large working sets respectively, and most of the working sets have cold data. However, *AppUpdate, Browser, SNS*, and *Map* have small size of working sets, each of which is updated frequently. *AppUpdate* and *SNS* are read-intensive workloads whereas others are write-intensive workloads.

C. Experimental Environments

In order to compare the performances of DFTL, CPM and K-CPM, a trace-driven FTL simulator is used in this paper. The latencies for page write, page read, and block erase operations are assumed to be 800 us, 60 us and 1.5 ms, respectively. The size of SRAM is 128 KB. The storage is initially aged. The 70% of the logical space is first written with sequential write requests, and then 20% of the aged logical space is overwritten with random write requests as the amount of total storage capacity.

In CPM and K-CPM, the size of one cluster is determined as 16 MB (16 blocks) considering the average working set size in TABLE II. Under the cluster size, one VTP can cover one cluster. One cluster can use up to 32 physical blocks (i.e., $N_{B\ max}^{C}$= 32) and K = 4. Among 128 KB of SRAM, 32 KB of memory is assigned for mapping tables other than PMT. The remaining 96 KB of SRAM is used for PMT. Only 96 KB of PMT entries can be loaded on demand into SRAM.

D. GC and Map Loading Overheads

Fig. 8 shows the GC overhead and map loading overhead of DFTL, CPM, and K-CPM. The overhead values are normalized by the total execution times. The overheads of CPM and K-CPM are less than those of DFTL due to the efficient hot and cold separation. In *AppUpdate* and *Map* workloads, the GC overheads of CPM are slightly worse than those of DFTL. This is because these workloads have small write requests with low temporal localities. Such workloads magnify the block utilization problem of CPM. However, due to the reduced map loading overhead, CPM shows better performance than DFTL at all workloads. K-CPM improves the GC overhead much more than CPM by enhancing the block utilization, whereas the map loading overhead is increased compared with CPM. During the experiments, the average value of N_α in K-CPM is 2.89, and K-InterGC is not invoked.

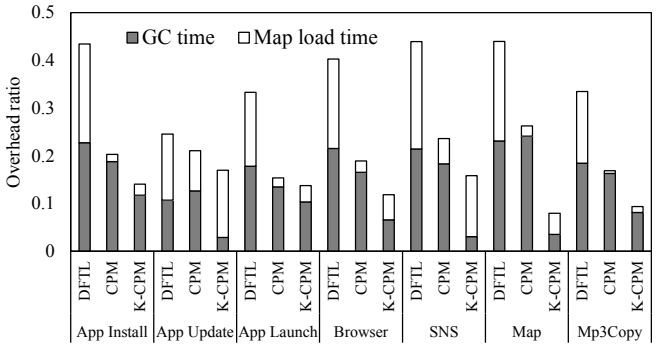

Fig. 8. Comparison on the GC and map loading overheads.

In the read-intensive workloads such as *AppUpdate* and *SNS*, the map loading overhead of K-CPM is much larger than CPM. This is because K-CPM should manage more additional mapping tables in RAM. Although DFTL has no additional mapping table, the low spatial locality among the valid pages in a victim block during GC makes a higher map loading cost.

Fig. 9 shows the ratios between IntraGC and InterGC in CPM and K-CPM. It also shows the number of block erase operations during GC. K-CPM invokes smaller number of GCs compared to CPM. CPM often selects InterGC whereas K-CPM rarely invokes InterGC. Since the distributed clusters can be used for IntraGC in K-CPM, there are more chances to select IntraGCs. The reduced number of block erase operations of K-CPM is beneficial to the limited lifetime of the flash storage.

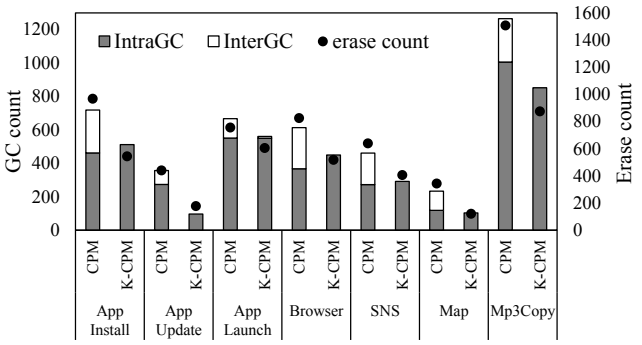

Fig. 9. GCs in CPM and K-CPM.

E. I/O Latency

Fig. 10 shows the cumulative distribution function (CDF) graphs on the I/O latencies during the executions of four workloads. There are significant differences at top 10% of I/O latencies. These long latencies are generated by GC and affect the overall performance. In the *AppInstall*, *AppLaunch*, and *Browser* workloads, the write latencies of CPM and K-CPM are shorter than those of DFTL. However, in the *Map* workload, CPM has a longer latency than DFTL. This results from the block utilization problem of CPM. The low block utilization invokes InterGCs frequently, which have higher costs than IntaGCs.

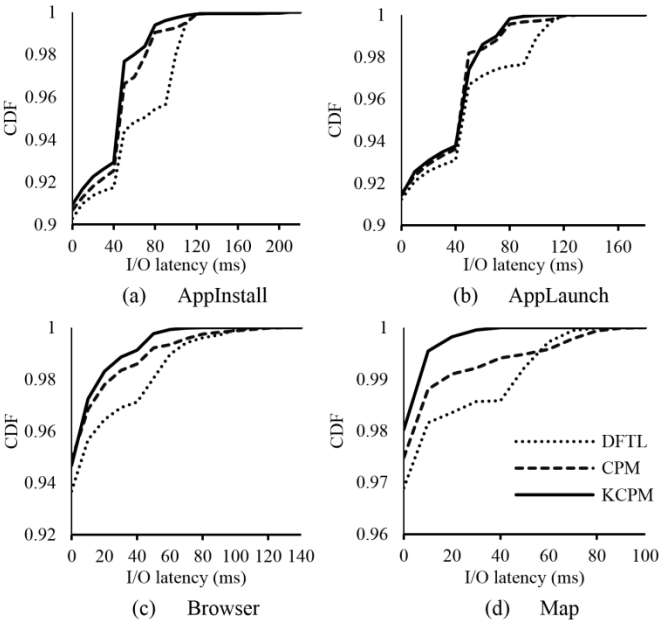

Fig. 10. CDF graph for I/O latencies.

F. Effects of Cluster Size and Associativity

Fig. 11 compares the GC and map loading overheads while varying S_C in CPM and K in K-CPM. Four different cluster sizes are used from 16 MB to 128 MB in CPM. Three different values of K are used from 2 to 8 while fixing S_C to 16 MB in K-CPM. As S_C grows, the number of VTPs required to cover a cluster will increase. The graph also shows the average remaining space, which represents the average number of remaining free pages when a GC is invoked. A low value of remaining space represents that the block utilization is high.

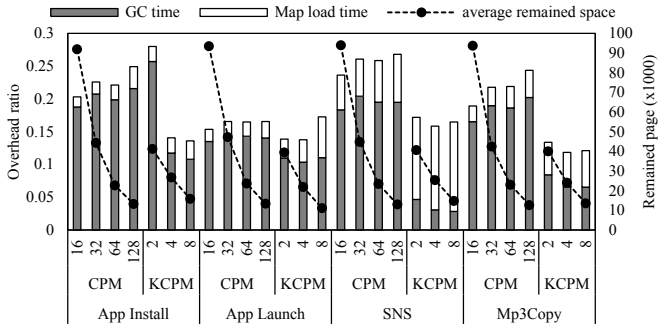

Fig. 11. The GC overhead and the remaining pages while varying S_C in CPM and N_a in K-CPM.

The graph shows that the average remaining space is reduced as S_C is enlarged in CPM. Therefore, a large cluster size can improve the block utilization. However, when the cluster is large, the GC cost increases since the hot and cold pages are mixed within a cluster. The map loading overhead also increases at larger cluster sizes. Therefore, a large cluster cannot be a solution on the low block utilization problem.

In K-CPM, when a larger value of K is used, the block utilization increases, and thus the performances are improved. However, if K is too large, the map loading overheads increase since the sizes of CAMT and CDMT are increased.

VII. Conclusion

The current page-level mapping FTLs show better performance compared to block-level mapping FTLs. However, they do not consider the spatial locality in storage workloads. This paper proposed the clustered page-level mapping (CPM) technique which can efficiently and effectively separate the hot and cold data based on the logical address. By allocating different physical blocks to different address ranges, the spatial locality within a physical block can be utilized. CPM also can reduce the map loading overhead during garbage collections compared with the previous demand map loading FTLs. However, the block utilization may deteriorate in CPM. To solve the problem, this paper proposed K-associative clustered page-level mapping (K-CPM) which allows multiple clusters to share a physical block. K-CPM can reduce the garbage collection cost by increasing the block utilizations.

References

[1] P. Desnoyers, "What systems researchers need to know about NAND flash," in Proc. USENIX Workshop on Hot Topics in Storage and Filesystems, 2013.

[2] T. Chung, D. Park, S. Park, D. Lee, S. Lee, and H. Song, "A survey of flash translation layer," *Journal of Systems Architecture*, 55(5-6), pp. 332-343, 2009.

[3] T. Shinohara, "Flash memory card with block memory address arrangement," United States Patent, no. 5,905,993, 1999.

[4] A. Ban, "Flash filesystem," United States Patent, No. 5,404,485, 1995.

[5] J. Kim, J. Kim, S. Noh, S. Min, and Y. Cho. "A space-efficient flash translation layer for compact flash systems," *IEEE Trans. on Consumer Electrons.*, 48(2), pp. 366–375, 2002.

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

[7] H. Cho, D. Shin, Y. Eom, "KAST: K-associative sector translation for NAND flash memory in real-time systems," in Proc. Conference on Design, Automation and Test in Europe, 2009.

[8] S. Lee, D. Shin, Y. Kim, and J. Kim, "LAST: locality-aware sector translation for NAND flash memory-based storage systems," in Proc. ACM International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability, pp. 36-42, 2008.

[9] D. Park, and D. Du, "Hot data identification for flash-based storage systems using multiple bloom filters," in Proc. IEEE Symposium on Mass Storage Systems and Technologies (MSST), 2011.

[10] S. Jung, Y. Lee, and Y. Song, "A process-aware hot/cold identification scheme for flash memory storage systems," *IEEE Trans. on Consumer Electrons.*, 56(2), pp. 339-347, 2010.

[11] M. Chiang, R. Chang, "Cleaning policies in mobile computers using flash memory", *Journal of Systems and Software,* 48(3), pp. 213-231, 1999.

[12] S. Im, and D. Shin, "ComboFTL: Improving performance and lifespan of MLC flash memory using SLC flash buffer," *Journal of Systems Architecture*, 56(12), pp. 641-653, 2010.

[13] P. Wu, Y. Chang, and T. Kuo, "A file-system-aware FTL design for flash-memory storage systems," in Proc. Conference on Design, Automation and Test in Europe, 2009.

[14] A. Mathur, M. Cao, S. Bhattacharya, A. Dilger, A. Tomas, L. Vivier, "The new ext4 filesystem: current status and future plans," Linux Symposium. Vol. 2. 2007.

[15] J. Kang, H. Jo, J. Kim, and J. Lee. "A superblock-based flash translation layer for NAND flash memory," in Proc. ACM and IEEE International Conference on Embedded Software, pp.161-170, 2006.

[16] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings," in Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 229-240, 2009.

Biographies

Hyukjoong Kim received the B.S. degree in electronics and electrical engineering, the M.S. degree in computer engineering from Sungkyunkwan University, Korea in 2011 and 2013. He is currently a Ph.D. Student in the department of IT convergence, Sungkyunkwan University. His research interests include embedded software, memory management, filesystem, 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 associate professor in the Department of Software 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 real-time systems. He is a member of the IEEE.