Anticipatory I/O Management for Clustered Flash Translation Layer in NAND Flash Memory

Kwanghee Park, Junsik Yang, Joon-Hyuk Chang, and Deok-Hwan Kim

Recently, NAND flash memory has emerged as a next generation storage device because it has several advantages, such as low power consumption, shock resistance, and so on. However, it is necessary to use a flash translation layer (FTL) to intermediate between NAND flash memory and conventional file systems because of the unique hardware characteristics of flash memory. This paper proposes a new clustered FTL (CFTL) that uses clustered hash tables and a two-level software cache technique. The CFTL can anticipate consecutive addresses from the host because the clustered hash table uses the locality of reference in a large address space. It also adaptively switches logical addresses to physical addresses in the flash memory by using block mapping, page mapping, and a two-level software cache technique. Furthermore, anticipatory I/O management using continuity counters and a prefetch scheme enables fast address translation. Experimental results show that the proposed address translation mechanism for CFTL provides better performance in address translation and memory space usage than the well-known NAND FTL (NFTL) and adaptive FTL (AFTL).

Keywords: FTL, AFTL, clustered hash table, prefetch, continuity counter.

Kwanghee Park (phone: + 82 32 860 7424, email: khpark@iesl.inha.ac.kr), Junsik Yang (email: juneseek@iesl.inha.ac.kr), Joon-Hyuk Chang (email: changjh@inha.ac.kr), and Deok-Hwan Kim (phone: + 82 32 860 7424, email: deokhwan@inha.ac.kr) are with the Department of Electronic Engineering, Inha University, Incheon, Rep. of Korea.

I. Introduction

Recently, mass NAND flash memory has emerged as a storage media alternative to the magnetic disk. Flash memory has several advantages such as faster access speed, lower power consumption, and better shock resistance than the traditional hard disk. Accordingly, it has been adopted in embedded applications, mobile consumer electronics, and various types of memories [1], [2].

As shown in Fig. 1, the basic architecture of the mobile storage for consumer electronics consists of flash memory chips, a dedicated controller, and a static random access memory (SRAM) for data I/O management. Therefore, the data structure and algorithm for the flash memory affects the performance of the mobile storage [3].

The flash memory contains several blocks, and each block consists of a fixed number of pages. A page comprises the user data area and the spare area. The user data area stores contents, and the spare area stores error correction codes and extra information, such as the logical block number used in the operating system and the block status information.

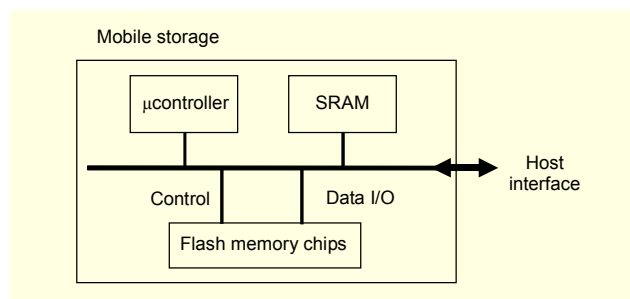Usually, there are two types of blocks: a large block

Fig. 1. Basic architecture of the mobile storage.

Table 1. Times and units of operations in NAND flash memory.

Operation	Unit size	Time
Read	512 B (random/sequential)	15 μs / 42 ns
	2 kB (random/sequential)	25 μs / 25 ns
Write	512 B / 2 kB	200 μs
Erase	16 kB	2 ms
	128 kB	1.5 ms

(128 kB) composed of 64 large pages (2 kB), and a small block (16 kB) composed of 32 small pages (512 B). Initially, the manufacturers supported a small block scheme, but as the capacity of flash memory increases, it needs to handle a large number of pages. However, it is hard to control a large number of pages because NAND flash memory contains an 8- or 16-bit microcontroller; therefore, the manufacturers now support a large block scheme as an alternative.

The read and write operations are performed in the unit of a page, whereas the erase operation is performed in the unit of a block. As shown in Table 1, the operations have various activity times [4], [5]. Because the unit of the read/write operations differs from that of the erase operation, flash memory incurs much overhead to update in place so that data is written to free space, and old versions of data become invalid. Furthermore, because the read operation in sequential access is faster than the read operation in random access, the spatial locality affects I/O performance.

The flash translation layer (FTL), a type of middleware, has been employed to connect the file system and the structure of NAND flash memory since the flash memory and a hard disk have different hardware characteristics. Specifically, NAND flash memory has a limited block life cycle of 100,000 times for the single-level-cell type and that of 10,000 times for the multi-level-cell type, respectively. Thus, FTL performs out-place update due to the characteristics of flash memory.

As the capacity of NAND flash memory continues to rapidly increase, the address translation mechanism of the FTL is becoming a critical design issue in terms of performance improvement and memory space requirement.

The adaptive FTL (AFTL), a well known flash memory management mechanism, combines two different granularities in address translation to exploit the advantages of page mapping and block mapping and enhances the address translation performance [6].

The AFTL uses hash tables for address translation. However, its fixed memory space overhead becomes 75% since it uses double pointers for each bucket. For example, the AFTL uses an additional 4 B single pointer for 8 B data in a coarse-grained

hash table. In a fine-grained hash table, a slot uses additional 8 B double pointers for 12 B data. Furthermore, the improvement ratio of the AFTL address translation time is insignificant even when the size of the fine-grained hash table becomes large.

To resolve this problem, we propose a clustered FTL (CFTL), which uses clustered hash tables as a data structure for the address translation and a two-level software cache technique. The proposed CFTL enhances the address translation performance by combining block mapping (a coarse-grained clustered hash table) and page mapping (two-level fine-grained clustered hash tables) and reducing memory space utilization. In addition, anticipatory I/O management using a continuity counter and a prefetch mechanism enhances the performance of consecutive address translation.

This paper is organized as follows. Section II describes the related works on other FTL algorithms. Section III describes the address translation mechanism for CFTL, and section IV presents experimental results. Finally, section V concludes the paper.

II. Related Works

1. Early Flash Translation Layer

The early FTL adopted a page-level address translation mechanism [7]. The early FTL represents a typical fine-grained FTL design.

Basically, the address translation table in an FTL is held in the random access memory (RAM) while the system is powered and the relationship between the logical block number used in the operating system and the physical block number of individual block or page is backed up in the spare area of the flash memory. The relationships are used to translate the logical block address (LBA) into the physical block address (PBA).

For example, as shown in Fig. 2, if we assume that a block consists of 8 pages, the LBA 8 is mapped to the PBA (8, 2) by the translation table. Its translation speed is fast because the address translation information is stored by page unit. However, the translation table becomes large because the address translation is performed by the unit of a page. When the size of the flash memory is 512 MB with a page size of 512 B, it requires 1,048,576 (512×1024×1024/512) entries of the address translation table to store the address translation information. When the size of an address information slot is 4 B, the early FTL requires 4 MB RAM space to convert from the LBAs to the PBAs. Moreover, it incurs a fixed overhead of 128 B per block 16 kB.
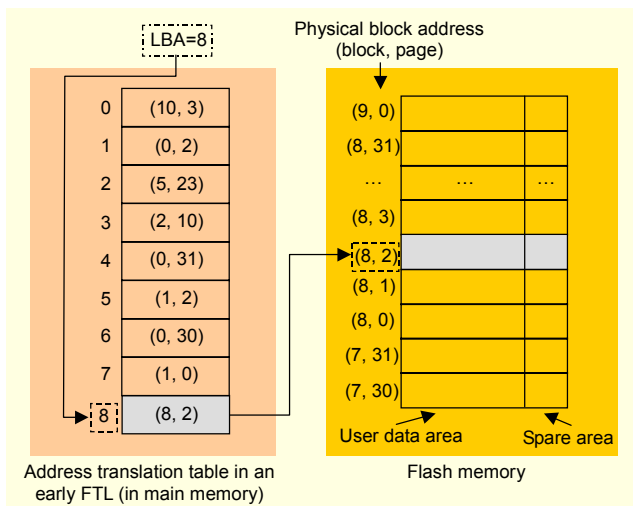
Fig. 2. Early flash translation layer.

Fig. 3. NAND flash translation layer.

2. NAND Flash Translation Layer

The NAND FTL (NFTL) designed by M-Systems adopts a block-level address translation mechanism. NFTL represents a typical coarse-grained flash translation layer design [8]-[11]. In NFTL, an LBA is divided into a virtual block address (VBA) and a block offset, where the VBA is the quotient when the LBA is divided by the number of pages in a block, and the block offset is the remainder. Each VBA is translated into a primary block and a replacement block, and the block offset is the order of the primary block.

When a write request is issued, data is written to the page corresponding to the block offset in the primary block. Because the subsequent write requests cannot overwrite the same pages in the primary block, a replacement block is needed to handle subsequent write requests, and the contents of the repeated write requests are sequentially written to the replacement block.

For example, as shown in Fig. 3, if we assume that a block consists of 8 pages and the LBA is 1,284, the LBA is divided into 8, and the VBA and the block offset become 160 and 3, respectively. Therefore, the corresponding primary PBA (PPBA) and the replacement PBA (RPBA) are 100 and 588, respectively. On the other hand, if the data is overwritten several times (as data A or B in the figure), it will be updated in the corresponding replacement block.

In NFTL, the translation table becomes small because the address translation is performed by the unit of a block. When the size of the flash memory is 512 MB with a page size of 512 B, it requires 32,768 (512×1024/16) entries of the address translation table to store the address translation information. When the size of an address information slot is 4 B, the NFTL requires 128 kB RAM space to convert from the LBAs to the PBAs.
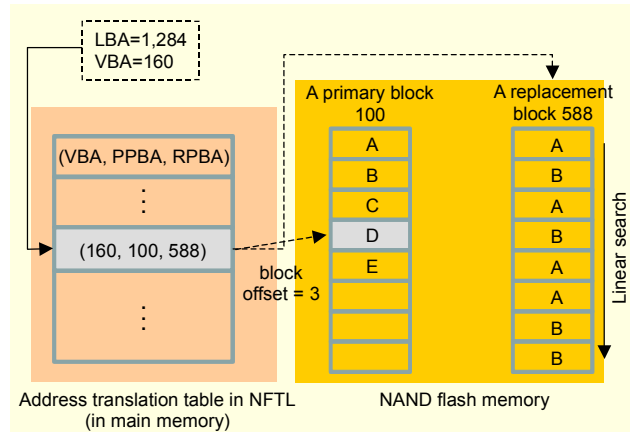
3. Adaptive Flash Translation Layer

The AFTL adopts a hybrid address translation mechanism that combines page-level address translation with block-level address translation.

With respect to a request for address translation, if AFTL finds a matching LBA in a fine-grained hash table, the matching address is translated by the table using a page-level address translation mechanism. Otherwise, the address is translated by a coarse-grained hash table using a block-level address translation mechanism.

The AFTL provides a switching policy to switch the most recently used mapping information to the fine-grained address translation table and, at the same time, switch the least recently used mapping information to the coarse-grained address translation table because of the limited resource of the fine-
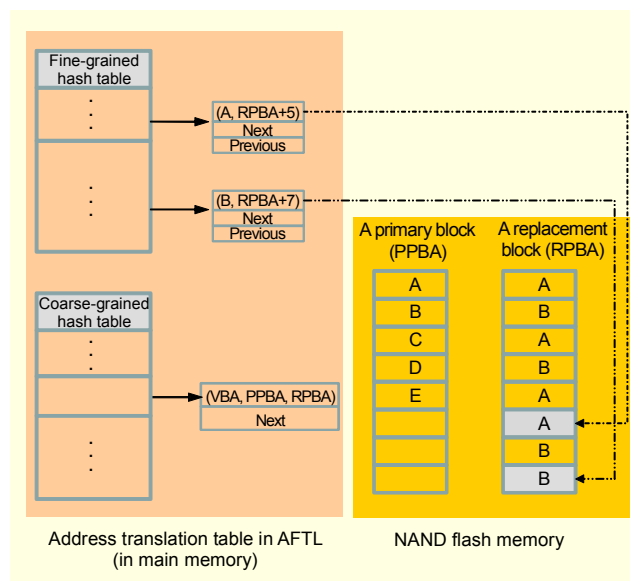

Fig. 4. Adaptive flash translation layer.

grained address translation table [6].

For example, as shown in Fig. 4, if the VBA "A" referenced in the RPBA is translated into PBA, the AFTL looks up a fine-grained hash table. If the VBA is found, the corresponding RPBA and offset are returned. Otherwise, the AFTL searches the coarse-grained hash table again.

Unlike NFTL, when the VBA exists in a fine-grained hash table, AFTL is able to translate without linearly searching for the RPBA since a fine-grained hash table stores the RPBA and offset, which enables page-level address translation.

III. Address Translation Mechanism for CFTL

1. Overview

CFTL consists of three clustered hash tables as shown in Fig. 5. First, two fine-grained clustered hash tables are designed as a two-level software cache. A short fine-grained clustered hash table stores the small number of slots with the most frequently referenced addresses, and a long fine-grained clustered hash table stores the large number of slots with frequently referenced addresses. Second, a coarse-grained clustered hash table has hit count and continuity counter fields as well as address information. The hit count field is used to select the frequently referenced slots, and the continuity counter field is used to distinguish whether addresses are consecutive or not.

There are several advantages of the address translation mechanism for CFTL. First, the structure of a clustered hash table which is adopted from the structure of a linear table and a hash table incurs less memory overhead than the traditional hash table. Also, it can preserve the locality of reference with respect to subblocks in a bucket because it groups subblocks with adjacent addresses into the same bucket. Second, the use of two-level fine-grained clustered hash tables adopted from a software cache technique reduces the miss penalty and decreases the access frequency of the coarse-grained clustered hash table, which entails relatively large address translation overhead. Third, the overhead of the address translation is reduced by using a continuity counter in the coarse-grained clustered hash table when a request for address translation with respect to consecutive VBAs occurs. Fourth, the anticipatory I/O management using a prefetch scheme enhances the performance of the address translation because it prefetches whole pages referenced in the bucket including the subblock matched with the requested LBA if the requested LBA corresponds with one of the subblocks in the fine-grained clustered hash tables.

When the system is powered up, the procedure of building the address translation table is as follows. Generally, LBAs are stored in the spare areas of pages. For that reason, CFTL reads all of the spare areas of the flash memory to fill the address translation table. The loaded logical addresses are inserted into corresponding slots in the coarse-grained clustered hash table in order. When CFTL has finished the address translation table setup, it waits for the I/O requests from the host. If a write request is received from the host, the data and the logical block address are written to the corresponding physical page and spare area, respectively. Subsequently, the corresponding slot of the address translation table is updated to the new physical address.

Figure 5 presents an overview of the address translation process in CFTL. When a request for address translation with respect to a specific LBA occurs, CFTL first searches the short fine-grained clustered hash table. If it finds a matching LBA in
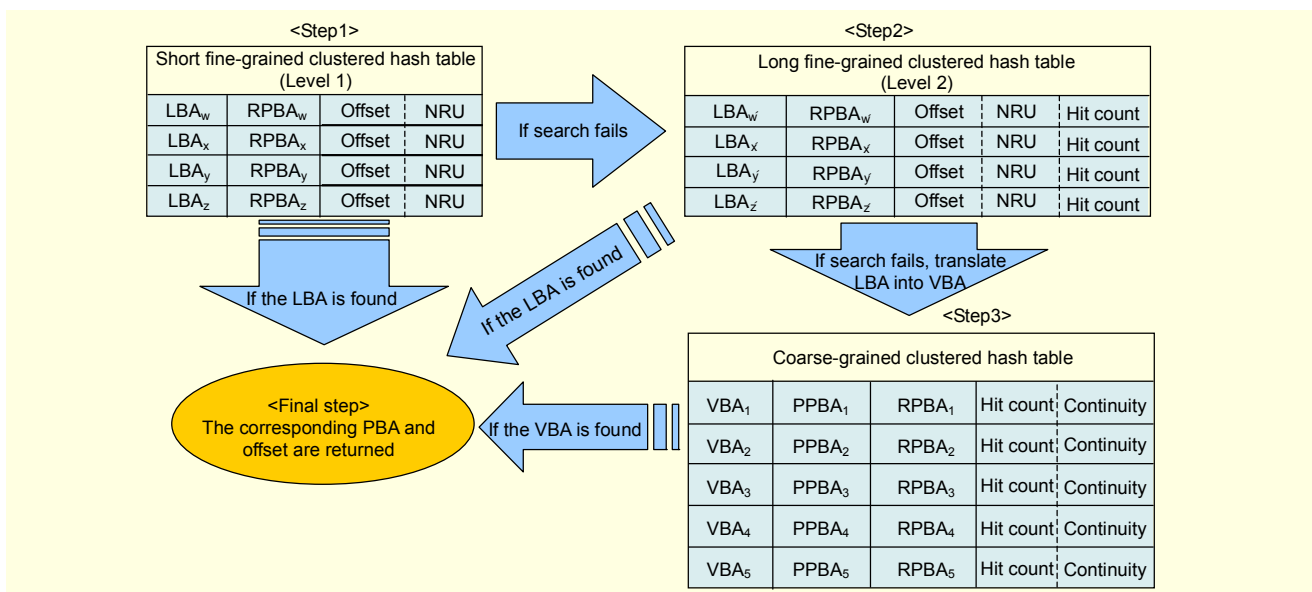

Fig. 5. Overview of address translation mechanism for CFTL.

the table, the address translation is completed and the referenced page using the corresponding PBA and offset is transferred to the host.

If search in the short fine-grained clustered hash table fails, it searches the long fine-grained clustered hash table. If it finds a matching LBA in the table, the address translation is completed, and the referenced page using the corresponding PBA and offset is transferred to the host. Otherwise, it searches the coarse-grained clustered hash table using a block mapping mechanism. It translates the LBA into a VBA with offset. It hashes the calculated VBA to find the bucket that stores PPBA and RPBA. If the corresponding PPBA and offset are valid, the data is transferred to the host. If not, CFTL searches the corresponding page of the RPBA in order, and then the corresponding data is transferred to the host.

2. Traditional Hashing Problems

Basically, most FTLs and the address translation scheme in virtual memory use hash tables to translate logical addresses into physical addresses. However, address translation schemes using hash tables have two disadvantages. The first problem is loss of locality of reference. The more the number of addresses increases according to the capacity of storage, the worse this problem becomes because the traditional hash table becomes sparse. Generally, the hash table consists of linked lists of buckets, and each bucket contains single or multiple pointer fields. The sparse hash table with linked-lists has many empty buckets; thus, a considerable amount of memory space is allocated in vain. The second problem is missing time due to hashing. Generally, the hash function makes use of mathematical function. If a hash function's computational complexity is low, its loss time is small but many collisions will occur. Otherwise, it can avoid collision but its performance will decrease.

3. Clustered Hash Table

Since the address translation time of a linear table increases proportionally to its size, it is not adaptable for mass flash memory. As shown in Fig. 6, the memory usage of a hash table increases more rapidly than that of a linear table since it wastes memory space by using double-linked lists including the previous pointer and the next pointer.

To resolve this problem, as shown in Fig. 7, we propose a clustered hash table as a data structure for address translation in the CFTL. This approach was originally used for a virtual memory paging method in Solaris 2.5, a commercial operating system on UltraSPARC-based computers of Sun Microsystems [12]. The clustered hash table is composed of buckets, which are sets of subblocks. The number of subblocks in a bucket is the subblock factor. The clustered hash table reduces the number of

pointers to $1/N$ by including N subblocks in the same bucket. It also reduces the average case search time to $O(\log N)$ during the address translation. Therefore, its memory usage for pointers is less than that of the hash table. Furthermore, since consecutive and concurrently used data is stored in the same bucket, the clustered hash table can preserve the locality of reference with respect to a set of address translation requests.

4. Continuity Counter

As shown in Fig. 8, the coarse-grained clustered hash table includes continuity counters to represent the number of continuous blocks. The continuity counter is set to C if the number of the continuous PPBA with respect to consecutive VBAs is C. This enables address translation without searching
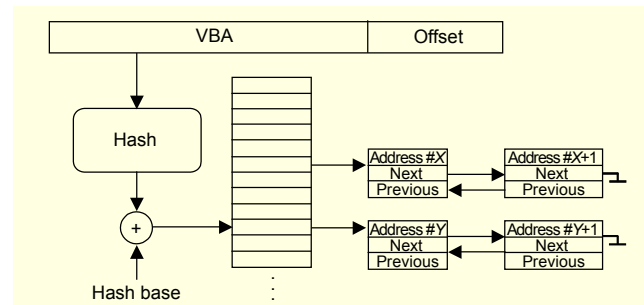

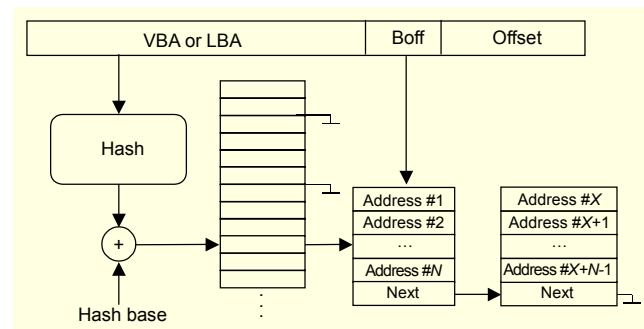Fig. 6. Structure of conventional hash table using doubly-linked lists.

Fig. 7. Structure of clustered hash table.

VBA	PPBA	RPBA	Continuity counter	Hit count
512	200	800	4	2
516	210	810	2	3
...				
518	220	820	0	1
519	221	750	3	4
Next pointer				

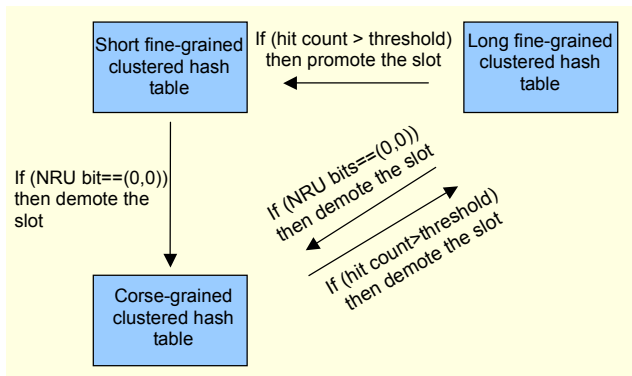Fig. 8. Example of a bucket in the coarse-grained clustered hash table.

Fig. 9. coarse-to-fine and long-to-short promotion policy; fine-to-coarse demotion policy.

the address translation tables.

For example, if the VBA is 512, the corresponding PPBA is 200, the continuity counter is 4, and the incremental value of the VBA is less than or equal to 4, then we can obtain the next PPBAs without searching the address translation tables. Furthermore, in that case, it is not necessary for the coarse-grained clustered hash table to have the address information of consecutive VBAs. Therefore, this technique reduces the memory space requirement of the coarse-grained clustered hash table.

5. Two-Level Fine-Grained Clustered Hash Tables

As shown in Fig. 9, two-level fine-grained clustered hash tables are implemented using the software cache technique, adopting the hardware translation lookaside buffer concept to enhance the performance of CFTL. Even though the use of two-level fine-grained clustered hash tables does not help reduce the access time to the fine-grained clustered hash tables due to the limitation of the software cache mechanism, it can reduce the miss penalty and decrease the access frequency of the coarse-grained clustered hash table. Therefore, it can reduce the address translation time since most of the address translation is performed in the fine-grained clustered hash tables.

For the two-level address translation tables, the data migration policy for a coarse-to-fine switch and a fine-to-coarse switch is needed. In the case of a coarse-to-fine switch, for example, if a subblock with a specific LBA in the coarse-grained clustered hash table is frequently accessed and its hit count is increased more than the threshold, the subblock is promoted to the long fine-grained clustered hash table and the hit count of this subblock is cleared. If the same subblock is increasingly accessed and its hit count is again more than threshold, it is again promoted to the short fine-grained clustered hash table. Meanwhile, the not recently used (NRU) page replacement method is used to demote unused subblocks to the coarse-grained clustered hash table. This is called a fine-to-coarse demotion policy.

6. Not Recently Used

NRU is a variation of the least recently used (LRU) approximation technique. It has less overhead than conventional LRU but provides similar performance [13].

NRU uses a system timer and determines unused slots periodically by using bit vectors, namely, a reference bit and a modify bit. Both the short fine-grained clustered hash table and the long fine-grained clustered hash table have NRU bits for the slot management. NRU bits are stored in the RAM, and they are initialized when the CFTL starts. For a fine-to-coarse switch, as shown in Table 2, two status bits are used to determine whether a subblock is demoted. For example, if the reference bit and modify bit of a subblock with a specific address are both zero, it is assumed that the subblock has not been used recently. The subblock is moved from the fine-grained clustered hash table to the coarse-grained clustered hash table, and the unused subblock is removed from the fine-grained clustered hash tables.

Table 2. Demotion table using NRU policy.

Reference bit	Modify bit	Description
0	0	Best slot to replace, so the slot is demoted
0	1	The slot will probably be updated again soon, so the slot stays in place
1	0	The slot will probably be read again soon, so the slot stays in place
1	1	The slot will frequently be used, so the slot stays in place

7. Prefetching Whole Pages from a Bucket

In general, FTL translates address and transfer data whenever an address translation request with respect to a specific LBA occurs. The prefetching method proposed in this paper prefetches whole pages referenced in a bucket, including the subblock matched with the requested LBA in the fine-grained clustered hash table.

The clustered hash table is a data structure for address translation which stores consecutive LBAs and corresponding PBAs in the same bucket. Therefore, if a read request of a page with a specific LBA occurs and a certain subblock in the fine-grained clustered hash table is matched with the LBA, physical random pages referenced in the subblocks adjacent to the matched subblock are prefetched. Even if subblock pages are not composed of consecutive physical pages, our proposed prefetching scheme enables fast random access on NAND
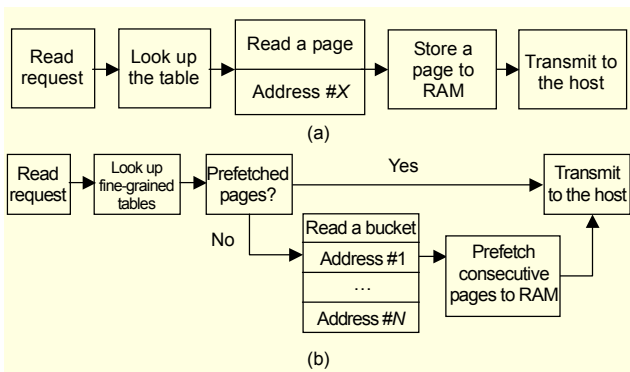
Fig. 10. Read operation (a) in traditional FTLs and (b) for prefetch pages in CFTL.

flash memory.

Figure 10(a) illustrates the sequence of a read operation in the traditional FTL. When a read command is requested from the host, the FTL searches the address translation table, reads the page, copies it to memory, and transfers it to the host. Figure 10(b) illustrates the read operation sequence of prefetching pages referenced by subblocks in a specific bucket in the clustered hash table. In this case, the CFTL prevents repetitive memory loading and repetition of command execution into the flash memory every time. Furthermore, the read time for the next read request can be reduced since multiple pages are prefetched.

IV. Performance Evaluation

To evaluate the proposed CFTL design, we compared CFTL with NFTL and AFTL in terms of address translation performance. For the experimental environment, we implemented the FTL schemes in a NAND flash memory simulator using the memory technology device open source of a LINUX kernel 2.6.17 environment [14]. We used a 512 MB NAND flash memory for comparison of the address translation time and the memory space requirement. In addition, we used 256, 512, 1024, 2048, 4096, and 8192 MB NAND flash memories for comparison of the block mapping table size.

To evaluate the memory space requirement, we set the ratio of the short fine-grained clustered hash table and the long fine-grained clustered hash table as 1:4. The number of maximum short fine-grained slots (MSFS) was set to 2,500 and the number of maximum long fine-grained slots (MLFS) was set to 10,000. Therefore, the number of maximum fine-grained slots (MFS) which is the sum of MSFS and MLFS was 12,500.

1. Address Translation Time

To evaluate the address translation time and the performance of the proposed method, we used two different block-level

Table 3. Block-level trace workloads.

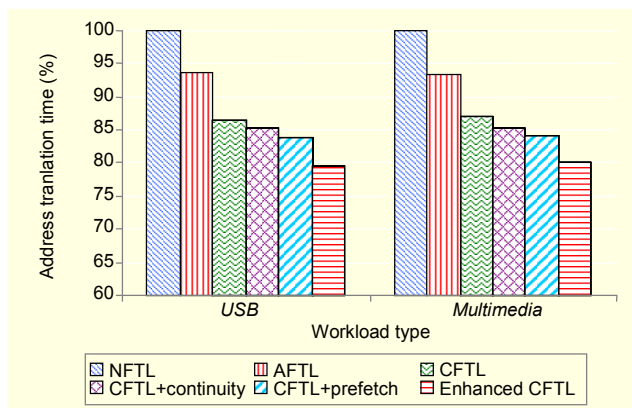| Name	Type	Duration	Year
USB	USB personal storage	5 days	2008
Multimedia	Mobile phone	5 days	2008

Fig. 11. Relative address translation times of FTLs per workload (NFTL=100%).

trace workloads as shown in Table 3. Each workload is a trace of I/O requests, and every entry is described by I/O time, I/O type, and real data. The first workload, *USB*, is a trace from the USB memory drive used for personal document files and source code files; the FAT32 file system resides on the USB memory drive. The second workload, *Multimedia*, is a trace from an external storage of the Motorola MS 700 mobile phone used for mobile multimedia files, such as photos, videos, e-books, audio files, and dictionary files. The FAT32 file system resides on the mobile phone. All workloads reflect real activities in daily life that totally fill the flash memory, randomly delete files, create files, and so on.

The test was repeated ten times under identical conditions. For this experiment, the subblock factor in the clustered hash tables was set to 8 and the threshold for coarse-to-fine and long-to-short promotion policies was set to 10.

As shown in Fig. 11, the address translation mechanism of pure CFTL [15] yields better performance and its address translation time is roughly 13% less than that of NFTL and about 7% less than that of AFTL.

In addition, in the case of adding anticipatory I/O management using continuity counters and the prefetch technique, the address translation time of enhanced CFTL is approximately 20% less than that of NFTL and about 14% less than that of AFTL.

2. Memory Space Requirement

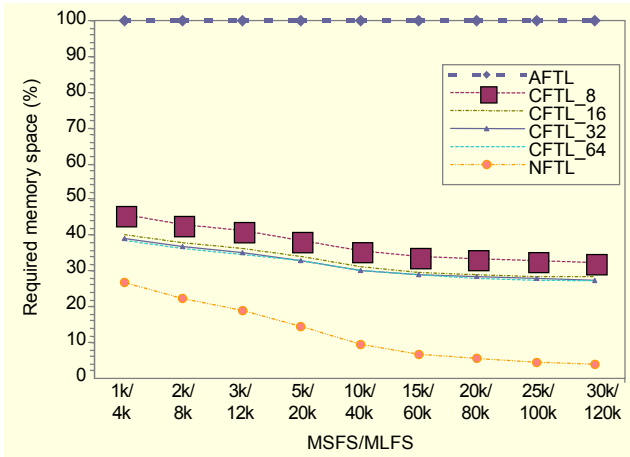Figure 12 shows the memory space requirements of NFTL,

Fig. 12. Relative memory space requirements of NFTL, AFTL, and CFTL (AFTL=100%).
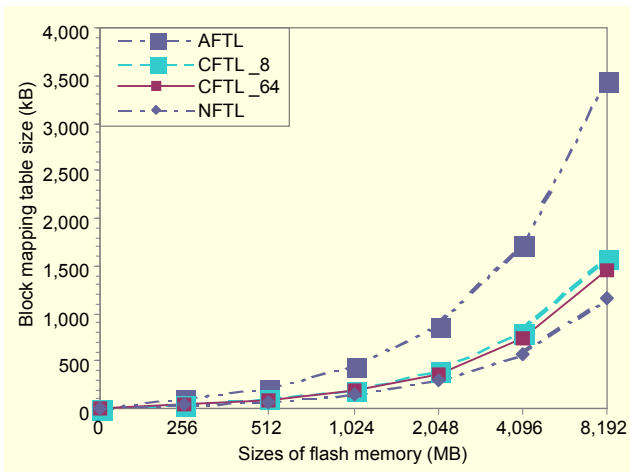

Fig. 13. Size of block mapping table in NFTL, AFTL, and CFTL.

AFTL, and CFTL with respect to various MFS values when the subblock factors of CFTL are 8, 16, 32, and 64, respectively.

Note that the memory space requirement of NFTL is constant with respect to various MFS values because there is no page mapping table in NFTL.

The result can be explained by the following facts. First, the size of the long fine-grained clustered hash table radically affects the performance in terms of the memory space requirement. This is because the long fine-grained clustered hash table is relatively larger than the short fine-grained clustered hash table and the coarse-grained clustered hash table. Second, the memory space requirement of AFTL sharply increases as the MFS increases. In contrast, that of CFTL gradually increases as the MFS increases, since CFTL reduces the memory space overhead by using clustered hash tables. Third, the ratio of the memory space requirement in CFTL is lower than that of AFTL by a maximum of 67.68%, 71.72%, 72.40%, and 72.73% when the subblock factors of CFTL are 8,

Table 4. Comparison of block mapping table size for small block

Flash memory size (MB)	NFTL (kB)	AFTL (kB)	CFTL_8 (kB)
256	64	192	80
512	128	384	160
1,024	256	768	320
2,048	512	1536	640
4,096	1,024	3,072	1,280
8,192	2,048	6,144	2,560

Table 5. Comparison of block mapping table size for large block.

Flash memory size (MB)	NFTL (kB)	AFTL (kB)	CFTL_8 (kB)
256	8	24	11
512	16	48	22
1,024	32	96	44
2,048	64	192	88
4,096	128	384	176
8,192	256	768	352

16, 32, and 64, respectively. The ratio was calculated based on the memory space requirement of AFTL. To evaluate the memory space requirements of the block mapping table in NFTL, AFTL, and CFTL, we used 256, 512, 1024, 2048, 4096, and 8192 MB NAND flash memories.

Figure 13 shows the size of the block mapping table in NFTL, AFTL, and CFTL using various subblock factors. We averaged the size of the block mapping tables for 16 kB and 128 kB block units. The results show that the rate of size increase of the block mapping table of CFTL is at most 72.73% lower than that of AFTL regardless of flash memory size, and the size of the block mapping table in CFTL is similar to that of NFTL.

V. Conclusion

The flash memory will continue to grow in popularity as its capacity increases and its cost decreases. Flash memory storage systems, such as Solid State Drive (SSD), will take a greater share of the market as an alternative mass storage system. Therefore, enhancement of the performance of address translation is imperative, and a small address translation table is similarly important.

The proposed CFTL is appropriate for a mobile mass storage system with NAND flash memory since its performance is better than that of traditional FTLs. Specifically, our technique reduces memory consumption by up to 72.73% and improves

speed by roughly 14% over the AFTL approach. It will be possible to adapt CFTL, to SSD or H-HDD if the hardware interface problems of respective storage devices are resolved.

References

[1] J. Kim et al., "A Space Efficient Flash Translation Layer for CompactFlash Systems," *IEEE Trans. Consumer Electronics*, vol. 48, no. 2, May 2002, pp. 366-375.

[2] S.L. Min and E.H. Nam, "Current Trends in Flash Memory Technology," *11th Asia and South Pacific Design Automation Conf.*, Yokohama, Japan, Jan. 2006.

[3] CompactFlash Association, "CompactFlash Specification Revision 4.1," http://www.compactflash.org.

[4] Samsung Electronics, "K9K8G08U1A", Data Sheet of NAND Flash Memory.

[5] Samsung Electronics, "K9F1208U0C", Data Sheet of NAND Flash Memory.

[6] C.H. Wu and T.W. Kuo, "An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems," *IEEE/ACM Int'l Conf. Computer Aided Design*, San Jose, CA, Nov. 2006.

[7] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," *Proc. the USENIX Technical Conf.*, Jan. 1995.

[8] Intel Corporation, *Understanding the Flash Translation Layer (FTL) Specification*.

[9] Intel Corporation, *Software Concerns of Implementing a Resident Flash Disk*.

[10] A. Ban, 1995. Flash file system. US patent 5,404,485. Filed March 8, 1993; Issued April 4, 1995; Assigned to M-Systems Flash Disk Pioneer Ltd., Tel Aviv, Israel.

[11] A. Ban, 1999. Flash file system optimized for page-mode flash technologies. US patent 5,937,425. Filed October 16, 1997; Issued August 10, 1999; Assigned to M-Systems Flash Disk Pioneer Ltd., Tel Aviv, Israel.

[12] M. Talluri, M.D. Hill, and Y.A. Khalidi, "A New Page Table for 64-bit Address Spaces," *Proc. the 15th ACM Symp. Operating Systems Principles*, Dec. 1995, pp. 184-200.

[13] A. Silberschatz, P.B. Galvin, and G. Gagne, *Operating System Concepts, 7th Edition*, John Wiley & Sons, Inc., 2005, pp. 315-326.

[14] MTD, "Memory Technology Device (MTD) subsystem for Linux," http://www.linux-mtd.infradead.org

[15] K.-H. Park and D.-H. Kim, "A Clustered Flash Translation for Mass Storage CompactFlash Systems," *IEEE Int'l Conf. Consumer Electronics*, Jan 12-14 2008, Las Vegas, U.S.A.

Kwanghee Park received the BS degree in computer and information technology from the Yong-In University, Korea, in 2005, and the MS degree in electronic engineering from Inha University, Incheon, Korea, in 2008. He is currently working toward the PhD degree in electronic engineering at Inha University, Incheon, Korea. His current research interests include intelligent algorithm in storage system, automotive software technology.

Junsik Yang received the BS degrees in computer and information engineering from Inha University, Incheon, Korea, in 2008 and He is working toward the MS degree in electronic engineering, at Inha University, Incheon, Korea. His research interests include hybrid storage systems and recommendation system.

Joon-Hyuk Chang received the BS degree in electronics engineering from Kyungpook National University, Daegu, Korea in 1998 and the MS and PhD degrees in electrical engineering from Seoul National University, Korea, in 2000 and 2004, respectively. From March 2000 to April 2005, he was with Netdus Corp., Seoul, as a chief engineer. From May 2004 to April 2005, he was with the University of California, Santa Barbara, in a postdoctoral position to work on adaptive signal processing and audio coding. In May 2005, he joined Korea Institute of Science and Technology, Seoul, as a Research Scientist to work on speech recognition. Currently, he is an assistant professor in the School of Electronic Engineering at Inha University, Incheon, Korea. His research interests are in speech coding, speech enhancement, speech recognition, audio coding, and adaptive signal processing.

Deok-Hwan Kim received the BS degree in computer science and statistics from Seoul National University, Korea in 1987 and the MS and PhD degrees in computer engineering from Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 1995 and 2003, respectively. From March 1987 to Feb. 1997, he was with LG Electronics, as a senior engineer. From Jan. 2004 to Feb. 2005, he was with University of Arizona, Tucson, in a postdoctoral position to work on multimedia systems and embedded software. Currently, he is an associate professor in the School of Electronic Engineering at Inha University, Incheon, Korea. His research interests include embedded systems, intelligent storage systems, multimedia system, and data mining.