An Adaptive Two-Level Management for the Flash Translation Layer in Embedded Systems

Chin-Hsien Wu
Department of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan, ROC
d90003@csie.ntu.edu.tw

Tei-Wei Kuo*
Department of Computer Science and Information Engineering,
Graduate Institute of Networking and Multimedia
National Taiwan University
Taipei, Taiwan, ROC
ktw@csie.ntu.edu.tw

ABSTRACT

While the capacity of flash-memory storage systems keeps increasing significantly, effective and efficient management of flash-memory space has become a critical design issue! Different granularities in space management impose different management costs and mapping efficiency. In this paper, we explore an address translation mechanism that can dynamically and adaptively switch between two granularities in the mapping of logical block addresses into physical block addresses in flash memory management. The objective is to provide good performance in address mapping and space utilization and, at the same time, to have the memory space requirements, and the garbage collection overhead under proper management. The experimental results show that the proposed adaptive mechanism could provide significant performance improvement over the well-known coarse-grained management mechanism NFTL (NAND Flash Translation Layer) over realistic workloads.

Categories and Subject Descriptors

C.3 [**Special-Purpose And Application-Based Systems**]: Real-time and embedded systems; D.4.2 [**Operating Systems**]: Storage Management: Secondary storage; B.3.2 [**Memory Structures**]: Mass storage

General Terms

Design, Performance, Algorithm

Keywords

Flash Memory, Flash Translation Layer, Storage Systems, Embedded Systems

*Supported in part by a research grant from the National Science Council under Grant NSC 93-2752-E-002-008-PAE and 94-2213-E-002-007, and a research grant from the Academia Sinica.

1. INTRODUCTION

Flash memory [1] is now among the top choices for storage media in embedded systems. Due to the very distinct characteristics of flash memory, the management of flash memory as a storage system is significantly different from those based on main memory and disks. In particular, flash memory is write-once such that updates to existing data on a page are only possible after an erase operation. Data must be written to free space, and the old versions of data are invalidated. Therefore, free space on flash memory could become low after a number of writes, and activities (i.e., garbage collection) in the recycling of available space on flash memory must be done from time to time. In order to resolve the write-once and the garbage collection problems for data on flash memory, a flash translation layer is proposed to emulate flash memory as block devices so that many existing file systems (e.g., FAT/DOS, EXT/EXT2, and NTFS, etc) could be built on them without any modifications.

There are currently two popular types of flash translation layers: FTL [12, 14, 15, 16] and NFTL [3, 13]. Because FTL is a fine-grained address translation mechanism, FTL can provide good address translation time, less garbage collection overhead, and high space utilization but with significant memory space in management. On the contrary, NFTL is a coarse-grained address translation mechanism such that the memory space requirements is small, but the address translation time, the garbage collection overhead, and the space utilization are worse than those of FTL. However, a fine-grained address translation mechanism (e.g., FTL) could not be applicable to resource-limited embedded systems due to its large memory space requirements, especially when the capacity of a flash-memory device is growing rapidly[1]. As a result, a coarse-grained address translation mechanism (e.g., NFTL) is proposed for large-scale flash-memory storage systems. In this paper, we propose an address translation mechanism that can dynamically and adaptively switch the mapping information of logical block addresses into physical block addresses between the fine-grained and the coarse-grained address translation mechanisms. The objective is to provide good performance in address mapping and space utilization and, at the same time, to have the memory space requirements, and the garbage collection overhead under proper management.

[1]32Gb NAND flash memory chips [18] are, in fact, under mass production at this time point.

The rest of this paper is organized as follows: Section 2 introduces an overview of flash memory. Related work and motivation are summarized in Section 3. Section 4 introduces an adaptive two-level management of a flash translation layer. Section 5 provides performance evaluation of the proposed method. Section 6 is the conclusion.

2. FLASH-MEMORY CHARACTERISTICS

A NAND [1, 18] flash memory chip consists of many blocks, and each block is of a fixed number of pages. A block is the smallest unit for erase operations, while reads and writes are done in pages. A page contains a user area and a spare area, where the user area is for the storage of a logical block, and the spare area stores ECC and other house-keeping information (i.e., LBA). The typical sizes of the user area and spare area of a page are 512B and 16B, respectively. The typical block size of a NAND flash memory chip is 16KB. Because flash memory is write-once, we do not overwrite data on each update. Instead, data are written to free space, and the old versions of data are invalidated (or considered as dead). The update strategy is called "out-place update". In other words, any existing data on flash memory could not be over-written (updated) unless it is erased. The pages store live data and dead data are called "valid pages" and "invalid pages", respectively.

After a certain number of page writes, free space on flash memory would become low. Activities that consist of a series of reads, writes, and erases with the intention to reclaim free spaces would then start. The activities are called "garbage collection" and considered as overheads in flash-memory management. The objective of garbage collection is to recycle invalid pages scattered over blocks so that they could become free pages after erasings. How to smartly choose blocks for erasing is the responsibility of a *block-recycling policy*. The block-recycling policy should try to minimize the overheads of garbage collection, due to live data copyings. Under the current technology, each flash-memory block has a limitation on the erase cycle count, e.g., 1 million (10^6). A worn-out block could suffer from frequent write errors. The "wear-levelling" policy should try to erase blocks over flash memory evenly so that a longer overall lifetime could be achieved.

3. RELATED WORK AND MOTIVATION

In recent years, issues on flash-memory management had drawn a lot of attention. Excellent research results and implementations have been reported on performance enhancement, especially on file systems, garbage collection, and system architecture designs [2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

3.1 FTL and NFTL

FTL [12, 14, 15, 16] adopts a page-level address translation mechanism (i.e., a fine-grained address translation). The translation table in Figure 1 is one kind of fine-grained address translation. The LBA "3" is mapped to the physical block address (PBA) "(0,6)" by the translation table. Note that LBA's are addresses of pages mentioned by the operating system, and each PBA has two parts, i.e., the residing block number and the page number in the block! On the contrary, NFTL [3, 13] adopts a block-level address mechanism (for coarse-grained address translation).

An LBA under NFTL is divided into a virtual block address and a block offset, where the virtual block address (VBA) is the quotient (i.e., that of the LBA divided by the number of pages in a block), and the block offset is the remainder of the division. A VBA can be translated to a (primary) physical block address by the block-level address translation. When a write request is issued, the content of the write request is written to the page with the corresponding block offset in the primary block. Since the following write requests can not overwrite the same pages in the primary block, a replacement block is needed to handle subsequent write requests, and the contents of the (overwritten) write requests are sequentially written to the replacement block. As shown in Figure 2, suppose that write requests to three LBA's $A = 100$, $B = 101$, and $F = 105$ are issued for 2 times, 8 times, and 1 time to a primary block and a replacement block, respectively, the most-recent contents of A, B, and F are also shown in the figure. NFTL must maintain a table in which each entry has a PBA of its primary block (and a PBA of its replacement block if needed).

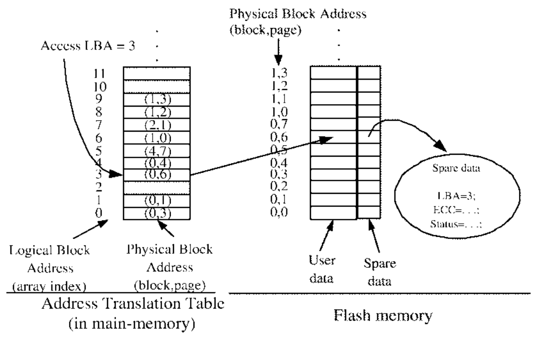


Figure 1: A RAM-resident Translation Table

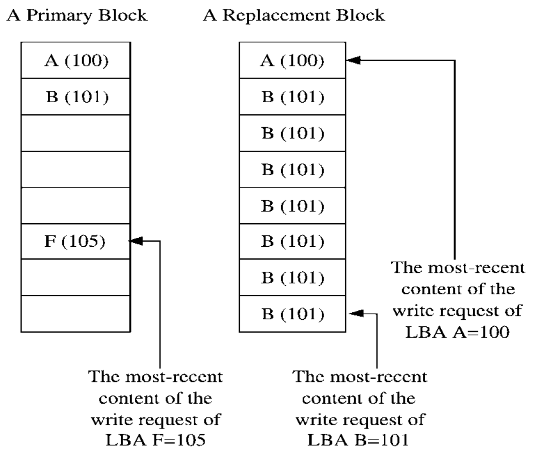

A Primary Block
A Replacement Block
A (100)
B (101)
F (105)
A (100)
B (101)
B (101)
B (101)
B (101)
B (101)
B (101)
B (101)
The most-recent content of the write request of LBA A=100
The most-recent content of the write request of LBA F=105
The most-recent content of the write request of LBA B=101

Figure 2: A Primary Block and a Replacement Block.

3.2 Comparison between FTL and NFTL

3.2.1 Memory Space Requirements

The main problem of FTL is on large memory space requirements for storing the address translation information because of its fine-grained address translation design (in the page level). For example, 256MB NAND flash memory with a page size of 512 bytes needs 524,288 (256*1024*1024/512) entries of the address translation table to store the address translation information. Let each entry require 4 bytes. The address translation information of FTL would require 2,048KB memory space. However, suppose that a block consists of 32 pages. NFTL would need roughly 64KB memory space to store 16,384 (256*1024/16) entries.

3.2.2 Address Translation Time

Because FTL adopts a fine-grained address translation mechanism, the translation from a given LBA to a PBA is very fast. On the other hand, NFTL could suffer from the slow address translation due to its coarse-grained address translation mechanism. When a read request is issued, NFTL must locate the most-recent content by searching the primary block and the replacement block whenever necessary. As shown in Figure 2, the address translation from LBA $F = 105$ to a corresponding PBA would incur a considerable access time of flash memory because the spare areas of pages in the replacement block might be searched entirely. Note that the read time of a spare area of flash memory is about $30\mu s$ [18].

3.2.3 Garbage Collection Overhead

When all of the pages of a replacement block are consumed, garbage collection should start to copy the valid pages of the corresponding primary block and the replacement block into a new primary block and then erase the two blocks for recycling. However, this copy overhead could be considerable because the number of valid pages could be equal to the capacity of the pages in a block. Compared to NFTL, FTL could choose a block that has the smallest valid pages to erase so that the garbage collection overhead could be less.

3.2.4 Space Utilization

Another problem for NFTL is the space utilization. When a replacement block is full (i.e., all free pages in the replacement block are exhausted), the replacement block and the corresponding primary block would be erased. However, some pages in the primary block could be free during erasing. As shown in Figure 2, the primary block still have 5 free pages during erasing. Compared to NFTL, FTL better utilizes the space of each block, and no free pages are left in a block before erasing.

3.3 Motivation

Table 1: The Summary of FTL and NFTL

	FTL	NFTL
Memory Space Requirements	Large	Small
Address Translation Time	Short	Long
Garbage Collection Overhead	Less	More
Space Utilization	High	Low

This research is motivated by the needs to seek a balance between fine-grained and coarse-grained address translation mechanisms. As shown in Table 1, FTL and NFTL are, in fact, complementary to each other. In this work, we shall propose an adaptive address translation mechanism that could provide good performance in address mapping and space utilization and, at the same time, to have the memory space requirements, and the garbage collection overhead under proper management.

4. AN ADAPTIVE TWO-LEVEL MANAGEMENT FOR THE FLASH TRANSLATION LAYER

4.1 Overview

We propose an adaptive flash translation layer called AFTL to exploit the advantages of the fine-grained and coarse-grained address translation mechanisms (referred to as the fine-grained AddrTM and the coarse-grained AddrTM hereafter). AFTL provides a block-device emulation of flash memory so that general file systems (e.g., FAT, NTFS, and ext2) can be built over it without modification. We propose an intelligent switching policy to switch the latest recently used mapping information (i.e., the mapping of LBA's into PBA's) to the fine-grained AddrTM, and at the same time, switch the least recently used mapping information to the coarse-grained AddrTM because of the limited resource of the fine-grained AddrTM.

4.2 Fine-Grained and Coarse-Grained Address Translation Mechanisms

The fine-grained AddrTM, that provides efficient mapping of the LBA of a page to its PBA, has a fine-grained hash table, where each table entry is a link list of fine-grained slots. Each fine-grained slot has two fields (LBA, PBA), where LBA and PBA are the LBA of a page and its corresponding PBA, respectively. Any given LBA is first hashed to a proper entry of the fine-grained hash table, and the corresponding link list is searched. If a matching is found, then the corresponding PBA is returned; otherwise, the LBA is given to the coarse-grained AddrTM for PBA look-up. In order to have the memory space requirements under control, the total number of fine-grained slots should be bounded (Please see the action in the switching of mapping information later in this section).

The coarse-grained AddrTM, that adopts an NFTL-like mechanism, is also associated with a coarse-grained hash table, where each table entry is a link list of coarse-grained slots. Each coarse-grained slot is a tuple (VBA, $PPBA$, $RPBA$), where VBA, $PPBA$, and $RPBA$ are the virtual block address (VBA), the PBA of a primary block of the VBA, and the PBA of a replacement block of the VBA (if needed), respectively. When an LBA of a page is received by the coarse-grained AddrTM, the corresponding virtual block address (VBA) is derived and hashed into the coarse-grained hash table. The corresponding link list of coarse-grained slots is then searched. The primary block and replacement block (if any) of the coarse-grained slot with the corresponding VBA is checked up in the NFTL way. In the following sections, we shall present the switching of the mapping information between the fine-grained AddrTM and the coarse-grained AddrTM.

4.2.1 Coarse-to-Fine Switches

When all of the pages in a replacement block in the coarse-grained AddrTM are used, the valid pages in the replacement block are identified, and the mapping information of the valid pages are moved to the fine-grained hash table by adding new fine-grained slots. They are considered as candidates for more efficient LBA mapping. As shown in

Figure 3, when all of the pages in the replacement block are used, the valid pages in the block are belonging to LBA's A and B. The mapping information of LBA's A and B are stored in two new fine-grained slots (A,$\delta_{cg}.RPBA+5$) and (B,$\delta_{cg}.RPBA+7$) and inserted to proper link lists of the fine-grained AddrTM. The mapping of LBA's A and B would be found through the fine-grained AddrTM thereafter. Here the residing block of LBA's A and B, i.e., $\delta_{cg}.RPBA$, is not recycled and is no longer used as the replacement block for the corresponding primary block. After the switches of mapping information, the PBA of the replacement block of the corresponding coarse-grained slot (i.e., RPBA) is nullified so that there is no replacement block for the corresponding primary block. Any subsequent write requests to LBA's C, D, or E in the example will be written to a new replacement block. Note that the corresponding coarse-grained slot and its primary block should be removed if all of the LBA's of the pages in its primary block appear in the replacement block. Different from many coarse-grained translation layers (e.g., NFTL), the replacement block and the corresponding primary block are not recycled immediately. Such two switches of mapping information are referred to as *coarse-to-fine switches* for the rest of this paper. The rationale behind the design of the coarse-to-fine switch procedure is as follows:

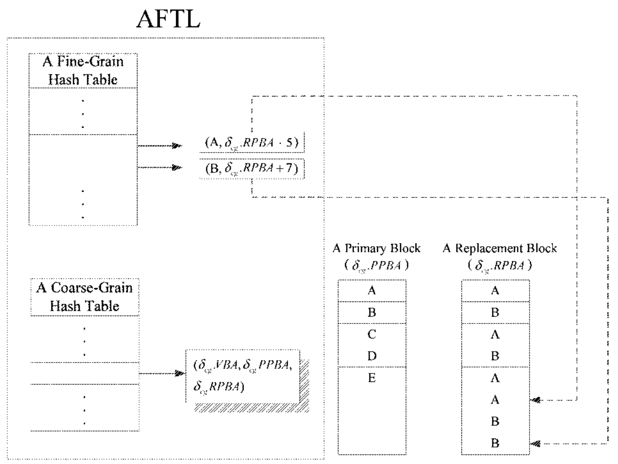

AFTL
A Fine-Grain Hash Table
(A, δ_{cg}.RPBA · 5)
(B, δ_{cg}.RPBA + 7)
A Coarse-Grain Hash Table
(δ_{cg}.VBA, δ_{cg}.PPBA, δ_{cg}.RPBA)
A Primary Block (δ_{cg}.PPBA)
A Replacement Block (δ_{cg}.RPBA)
A
B
C
D
E
A
B
A
B
A
A
B
B

Figure 3: The Moving of Frequent Used Mapping Information to the Fine-Grained Hash Table

- Because the valid pages in a replacement block are potentially belonging to frequently used LBA's, i.e., those with hot data, they would become invalid soon. The delayed recycling of any replacement block under AFTL might reduce the potential number of valid data copyings and the garbage collection overhead.
- Because of the delayed recycling of any primary block under AFTL, free pages of a primary block might be used in the future. As a result, the space utilization might be better.
- Because the valid pages in a replacement block are potentially belonging to frequently used LBA's, the moving of their mapping information to the fine-grained hash table might improve the address translation performance. On the other hand, the address translation performance for LBA's residing in the corresponding primary block is also improved because the RPBA field of the corresponding coarse-grained slot is nullified (such that there is no needs to scan the replacement block).

4.2.2 Fine-to-Coarse Switches

Because the number of the fine-grained slots is limited, some least recently used mapping information of fine-grained slots should be moved to the coarse-grained hash table whenever necessary. In order to maintain the access time information of fine-grained slots, an LRU double-link list is maintained, where each fine-grained slot is associated with two LRU pointers for the maintenance of the LRU double-link list.

Given an LRU fine-grained slot (LBA, PBA), the slot is first removed from the corresponding link list of the fine-grained hash table. The corresponding VBA of LBA is then derived, and the data stored in the page with the given PBA is copied to the primary or replacement block of the corresponding coarse-grained slot, as defined by the coarse-grained AddrTM. The original page with the given PBA is invalidated. If there does not exist any corresponding coarse-grained slot, a new coarse-grained slot is created, and its primary block is allocated. Such a switch of mapping information introduces valid-page-copying overhead due to the differences of address translation mechanisms, and it is referred to as a *fine-to-coarse switch* for the rest of this paper.

For example, let a fine-grained slot ($LBA = 1234$,$PBA = 5678$) be an LRU slot to be moved to a coarse-grained AddrTM. The fine-grained slot is removed from the corresponding link list of the fine-grained hash table. The coarse-grained AddrTM first derives the VBA and the block offset of LBA 1234. Let δ_{cg} be the coarse-grained slot that would store the mapping information of LBA 1234. Suppose that each block contains 32 pages. The block offset of LBA 1234 is 18 because $1234\ \%\ 32 = 18$. If the page with the block offset 18 of the primary block ($\delta_{cg}.PPBA$) is free, then the data in the page of PBA 5678 is copied to the page in the primary block; otherwise, the page of PBA 5678 is copied to the first free page in the replacement block ($\delta_{cg}.RPBA$). If there is no free page in the replacement block, then the coarse-grained AddrTM should first rearrange pages on the primary block and replacement block as defined previously (Please see NFTL in Section 3). The page of PBA 5678 is then copied to the proper page in the primary block as defined.

Since the number of the fine-grained slots is limited, coarse-to-fine switches would introduce fine-to-coarse switches and overhead in valid page copying. One way to manage the overhead in valid page copying is to stop any coarse-to-fine switch when some frequency bound in coarse-to-fine switches is reached. It could be done by rearranging pages on the primary block and replacement block as defined for NFTL in Section 3, instead of moving some mapping information belonging to valid pages in the replacement block to the fine-grained hash table. In the experiments, we set up different thresholds to control the frequency of coarse-to-fine switches and to observe the performance issues versus the overhead.

5. PERFORMANCE EVALUATION

5.1 Experimental Setup and Performance Metrics

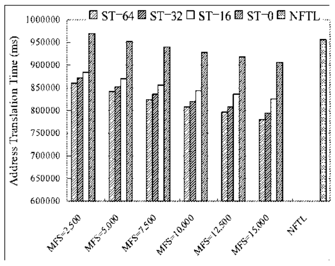
Table 2: Trace Characteristics

CPU	Intel Celeron 750 MHz
RAM	320 MB
Operating System	Windows XP
File System	NTFS
Storage Capacity	20GB
Applications	Web Applications, E-mail Clients, MP3 Player, MSN Messenger, Word, Excel, Power Point, Media Player, Programming, and Virtual Memory Activities
Duration	1 week
Total Write/ Read Requests	13,198,805 / 2,797,996 sectors
Different LBA's	1,669,228

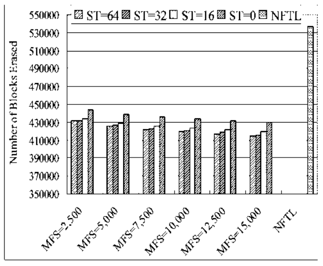
The characteristics of the experiment trace over a 20GB disk is summarized in Table 2. In the trace, there were 13,198,805 and 2,797,996 sectors that were written and read, respectively, where each sector was of 512B. We must point out that there were 1,669,228 different LBA's that were accessed. The trace shows that many written data had spatial locality, where each LBA was written for 7.9 times averagely. During the collection of the trace, real applications were executed to have realistic workloads in daily life. In the experiments, AFTL was emulated over an 20GB NAND type flash memory, and the experiments were conducted with AFTL and NFTL by running the collected trace. The block size, the page size, and the size of the spare area of each page were 16KB, 512B, and 16B, respectively, where there were 32 pages per block. The maximum number of fine-grained slots was controlled by a parameter MFS. The larger the MFS value is, the more the memory space to store the fine-grained slots, where each fine-grained slot occupied 20B. The switching threshold was set by a parameter ST that controls the frequency of coarse-to-fine switches. In the experiments, ST ranged from 64, 32, 16, to 0. If ST is not 0, then AFTL could have n/ST coarse-to-fine switches at most, where n is the total number of the requests. If ST is 0, then there is no constraint on the number of coarse-to-fine switches. Note that NFTL (e.g., a traditional coarse-grained AddrTM) is adopted for many large-scale NAND flash storage systems because of its reasonable memory space requirements. In the experiments, we have NFTL being a baseline to evaluate the performance of AFTL.

5.2 Memory Space Requirements

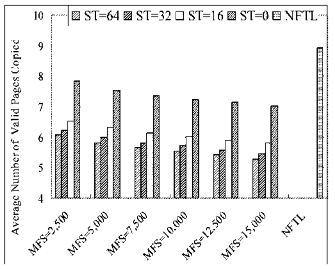
If a fine-grained (or page-level) address translation mechanism, e.g., the fine-grained AddrTM, was the only adopted mechanism to run the trace, then there would be 1,669,228 fine-grained slots. Even when a fine-grained slot occupied 4B, there would be 6.37MB (i.e., 1,669,228*4B) memory space needed for address translation. Such memory space requirements would make a fine-grained address translation mechanism (e.g., FTL) being infeasible to large-scale NAND flash-memory storage systems. In order to constrain the memory space requirements, the maximum number of fine-grained slots had to be bounded, where the number of coarse-grained slots was not restricted. In the experiments, MFS ranged from 2,500, 5,000, 7,500, 10,000, 12,500, to 15,000, and the memory space requirements for fine-grained slots ranged from 49K, 98K, 147K, 196K, 245K, to 293K. Compared to a coarse-grained address translation mechanism, e.g., the coarse-grained AddrTM or NFTL, all of the coarse-grained slots would need 741K memory space, when each coarse-grained slot occupied 12B. Since AFTL adopts a two-level address translation mechanism, i.e., fine-grained and coarse-grained AddrTM's, the increased ratio of memory space requirements ranged from 6.6% (i.e., 49K/741k), 13.2% (i.e., 98K/741K), 19.8% (i.e., 147K/741K), 26.5% (i.e., 196K /741K), 33.1% (i.e., 245K/741K), and 39.5% (i.e., 293K/741K) for different settings of MFS, compared to that of NFTL.



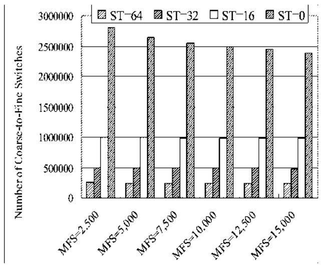
(a) Address Translation Performance



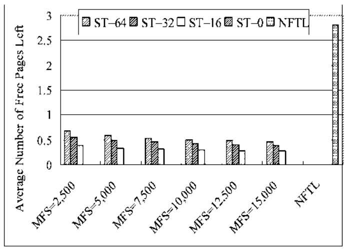
(b) The Number of Erased blocks



(c) The Average Number of Valid Page Copyings per Block Erasing



(d) The Number of Coarse-to-Fine Switches



(e) The Average Number of Free Pages Left per Block Erasing

Figure 4: Experimental Results

5.3 Address Translation Time

Figure 4.(a) shows the performance of AFTL under different numbers of fine-grained slots, i.e., MFS, and the coarse-to-fine switching threshold, i.e., n/ST (where n was the total number of requests so far). The larger the MFS value was, the smaller the address translation time. It was because a larger number of address translations went through the fine-grained AddrTM when MFS was larger. When MFS was 15,000 and ST was 64, the improvement ratio could be 18.4%, compared to NFTL. On the other hand, the smaller the ST value was, the larger the address translation time. It was because a small ST value encouraged a significant number of coarse-to-fine switches. As a result, the mapping information of LBA's rotated quickly between the fine-grained AddrTM and the coarse-grained AddrTM so that fine-grained slots were not used effectively before they faced fine-to-coarse switches again. The problem became more serious when MFS was not large enough to contain enough mapping information for frequently accessed LBA's. In summary, MFS and ST should have sufficiently large values to have good system performance.

5.4 Garbage Collection Overhead

The objective of garbage collection is to recycle the space occupied by invalid pages scattered over blocks. Before erasing a to-be-recycled block, data of all of the valid pages of the block must be copied to other free pages. Figures 4.(b) and 4.(c) show that the number of erased blocks and the average number of valid page copyings per block erasing are much lower than the corresponding numbers of NFTL (i.e., 53,7000 erased blocks and 8.9 valid page copyings per block erasing), respectively. It was due to the fact that coarse-to-fine switches sent the mapping information of frequently accesses data under the management of fine-grained AddrTM and avoid immediately recycling of their primary and replacement blocks and related valid data copyings, i.e., their block erasing. Figure 4.(d) shows that the number of coarse-to-fine switches increased when ST became small. It was because a small ST value encouraged more coarse-to-fine switches.

5.5 Space Utilization

The space utilization was defined as the ratio of the number of used pages to the maximum number of pages in a block. It also denotes the number of free pages in a block. The figure was important because it reflected the effectiveness in the space utilization under a coarse-grained address translation mechanism, such as NFTL, especially when we had rearrangement and recycling of primary and replacement blocks. Figure 4.(e) shows the average number of free pages per block erasing, where the average number was almost 0 when $ST = 0$. As shown in the experiments, the number of free pages per block erasing under NTFL was around 2.816996 so that its space utilization ratio was 91.2% (i.e., (32-2.816996)/32). AFTL was better with all of the ST values. When $ST = 0$, the space utilization was almost 100%. Even when $ST = 64$ and $MFS = 2500$, the space utilization of AFTL was still as high as 97.9%. In general, a small ST value had a better space utilization because coarse-to-fine switches were encouraged such that free pages in primary blocks had higher chances to be used before their block erasings.

6. CONCLUSION

This paper proposes an adaptive two-level management design of a flash translation layer, called AFTL, to exploit the advantages of the fine-grained AddrTM and the coarse-grained AddrTM. AFTL can dynamically and adaptively switch the mapping information of logical block addresses between the fine-grained and the coarse-grained address translation mechanisms. In the paper, we present our switch operations to handle the mapping information of logical block addresses, and an intelligent switching policy is presented to improve the system performance in address mapping and space utilization. The capability of AFTL was evaluated by a series of experiments under realistic workloads. It shows that AFTL does provide good performance in address mapping and space utilization and, at the same time, have the memory space requirements, and the garbage collection overhead under proper management.

For future research, we should further explore different application characteristics and different workloads in flash memory management. More research and tool designs in the customization of flash-memory storage systems for different embedded application systems might prove being very rewarding.

7. REFERENCES

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to Flash Memory," Proceedings of The IEEE, Vol. 91, No. 4, April 2003.

[2] L. P. Chang and T. W. Kuo, "An Adaptive Stripping Architecture for Flash Memory Storage Systems of Embedded Systems," IEEE Eighth Real-Time and Embedded Technology and Applications Symposium (RTAS), San Jose, USA, Sept 2002.

[3] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho, "A Space-Efficient Flash Translation Layer for Compact-Flash Systems," IEEE Transactions on Consumer Electronics, Vol. 48, No. 2, MAY 2002.

[4] M. Wu, and W. Zwaenepoel, "eNVy: A Non-Volatile, Main Memory Storage System," Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 1994), 1994.

[5] A. Kawaguchi, S. Nishioka, and H. Motoda, "A Flash-Memory Based File System," USENIX Technical Conference on Unix and Advanced Computing Systems, 1995 .

[6] H. J. Kim and S. G. Lee, "A New Flash Memory Management for Flash Storage System," Twenty-Third Annual International Computer Software and Applications Conference October 25 - 26, 1999 Phoenix, Arizona.

[7] C. H. Wu, L. P. Chang, and T. W. Kuo, "An Efficient B-Tree Layer for Flash-Memory Storage Systems," accepted and will appear in ACM Transactions on Embedded Computing Systems (TECS).

[8] C. H. Wu, T. W. Kuo, and L. P. Chang, "The Design of Efficient Initialization and Crash Recovery for Log-based File Systems over Flash Memory," accepted and will appear in ACM Transactions on Storage (TOS).

[9] C. H. Wu, L. P. Chang, and T. W. Kuo, "An Efficient R-Tree Implementation over Flash-Memory Storage Systems," The 11th International Symposium on Advances in Geographic Information Systems (ACM-GIS 2003).

[10] C. H. Wu, T. W. Kuo, and C. L. Yang, "Energy-Efficient Flash-Memory Storage Systems with Interrupt-Emulation Mechanism," accepted and to appear in the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis, Stockholm, Sweden, September, 2004.

[11] C. H. Wu, T. W. Kuo, and C. L. Yang, "A Space-Efficient Caching Mechanism for Flash-Memory Address Translation," The 9th IEEE International Symposium on Object and component-oriented Real-time distributed Computing (ISORC), Gyeongju, Korea, April, 2006.

[12] U.S. Pat. No. 5,404,485 "FLASH FILE SYSTEM"

[13] U.S. Pat. No. 5,937,425 "FLASH FILE SYSTEM OPTIMIZED FOR PAGE-MODE FLASH TECHNOLOGIES"

[14] Intel Corporation, "Understanding the Flash Translation Layer(FTL) Specification".

[15] Intel Corporation, "Software Concerns of Implementing a Resident Flash Disk".

[16] Intel Corporation, "FTL Logger Exchanging Data with FTL Systems".

[17] Intel Corporation, "LFS File Manager Software: LFM".

[18] Samsung Electronics. NAND flash-memory datasheet and SmartMedia data book, 2006.