New Techniques for Real-Time FAT File System in Mobile Multimedia Devices

Sunhwa Park and Seong-Young Ohm

Abstract — *Flash memory has become the most important storage media in the mobile multimedia products such as MP3 players, mobile phones, and digital cameras. Most mobile multimedia devices, however, use one of the conventional FAT file systems slightly modified for flash memory environments. Our analysis shows that theses file systems have some restriction in recording a live multimedia stream stably because they have irregular write response times.*

In this paper, we have considered the problems of the existing FAT file system and propose two new techniques to solve the problems. The first technique called Sector Reservation method reduces internal overhead effectively. And the other method called ACPA avoids the periodic cluster allocation of the conventional FAT file system and removes the frequent modifications on the file allocation table in the FAT file system.

To evaluate our new techniques, we implemented a prototype Real-Time FAT file system on ARM9 board with our two novel techniques. The experimental results show that our system achieves our goal successfully in that its write response times are very deterministic and more uniform[1].

Index Terms — Flash memory, FAT file system, FTL, Mobile multimedia device.

I. INTRODUCTION

One of the distinct trends in recent years is the growth of mobile multimedia devices. Tiny MP3 players and digital cameras are replacing CD players and film cameras, respectively. In addition, the mobile multimedia products are also being equipped with various multimedia functionalities. For example, recent mobile phones can take photos, play/record digital music as well as moving pictures, and even provide DMB (Digital Multimedia Broadcasting) TV and wireless internet services.

In these products, the role of the mobile storage is very important. Flash memory has many characteristics, such as low power consumption, shock resistance, light weight, small size, fast read/write speed, and large capacity, which are good for the mobile multimedia products. Therefore, most mobile multimedia products choose flash memories as their internal and/or external storages.

Meanwhile, as the capacity of the mobile storage increases up to several Giga bytes, the file system has been required to manage the storage effectively [1]. Many mobile multimedia products use the FAT file system [1][2]. It has a long history, a simple architecture, and a higher compatibility with other systems. The FAT file system, however, is not originally designed for flash memory, but for magnetic storages such as floppy diskettes and hard disks. Therefore, an internal software layer called FTL (Flash Translation Layer) [3] is added in the FAT file system for flash memory. FTL hides unique characteristics of the flash memory and emulates a hard disk with flash memory implicitly.

In addition, to support the sophisticated multimedia services, the recent mobile multimedia products are required to record a live multimedia stream stably. To record a multimedia stream without loosing data, its file system as well as the mobile storage should satisfy the time restriction of the requested write operation. Otherwise, RAM buffers, which are used to store multimedia stream data, may overflow and loose some part of the live stream.

In this paper, we assume that double buffers with the limited capacity are used for the transmission of multimedia streams, and focus on how to record them stably into the flash memory in the FAT file system with internal FTL.

Fig. 1 depicts how the multimedia stream is recorded. Multimedia data generation rate (R) is given to the system, and the buffer size (B) can be controlled by a developer. Time allowed for writing B bytes is T_w. To avoid buffer overflow, all the data in one buffer are to be written to the mobile storage within T_w, that is, before the other buffer becomes full.

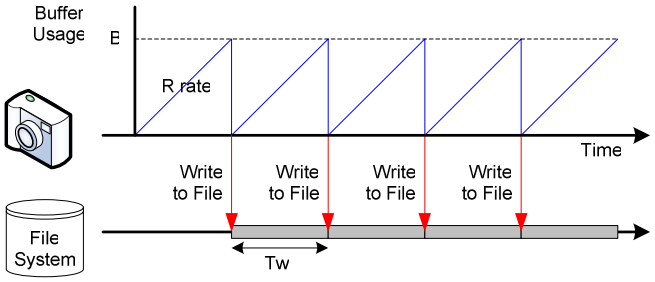

Fig. 1. Multimedia stream recording scheme: R means data transmission rate of multimedia stream, B buffer size, and Tw time restriction given for writing B bytes.

The existing FAT file systems for mobile devices, however, do not satisfy the real-time requirements. Our analysis on these systems shows that such the restriction in real-time application comes from two obstacles.

The first one is the internal overhead in FTL. For example, a simple sector write operation in the file system level may cause

[1] This work was supported by Seoul Women's University Bahrom Research Grant.

Sunhwa Park is with the Computer Science Department, Seoul Women's University, Korea (e-mail: bban@swu.ac.kr).

Seong-Yong Ohm is with the Computer Science Department, Seoul Women's University, Korea (e-mail: osy@swu.ac.kr).

Manuscript received January 15, 2006

several internal operations such as erasing the whole block and/or copying multiple pages in FTL level. Such the implicit operations in FTL may take a relatively long time, making it hard to predict precisely the response time of FTL.

The other one is the periodic cluster allocation policy adapted in the FAT file system. As the size of a file is increasing, a new cluster is allocated to the file one by one and FAT table is modified accordingly. Generally frequent FAT table modification causes much data updates which take relatively long time in FTL. This is the reason why FAT file system produces irregular response times, especially for the large size multimedia data.

In this paper, we present two novel techniques to overcome these problems: *Sector Reservation method* and *ACPA (All Clusters Pre-Allocation) method*. Sector reservation method reduces the possibility of the occurrence of some unexpected time consuming operations during recording the huge multimedia stream, and ACPA method avoids the periodic cluster allocations and frequent FAT entry modifications during recording the multimedia data.

To evaluate our new techniques, we modified one existing FAT file system, and implemented a prototype Real-Time FAT file system on ARM9 board with our two techniques. To demonstrate the performance of our approach, we measured write response times of our new system for sequential write requests, and compared them with those of an existing FAT file system. The experimental results show that our system achieves our goal successfully and its write response times are predictable and uniform for the sequential write requests.

The rest of this paper is organized as follows. Section II describes related work. Section III describes the background of our work for better understanding. Section IV describes our new techniques for Real-Time FAT file system in detail and explains how they can achieve the goal. Section V shows some of our experimental results and demonstrates the performance of our approach. Section VI concludes with a summary and some future work.

II. RELATED WORK

Several studies on flash memory system have been performed.

eNVy architecture [4] proposed by M. Wu and W. Zwaenepoel regards flash memory as a non-volatile main memory. In this research, they use NOR flash memory, a hardware MMU, a cleaning processor, and battery backuped SRAM's. They use their own sector mapping algorithm and propose several garbage collection policies.

A. Kwaguchi et al. tries to regard flash memory as storage for file system [5]. To adapt the existing file systems to flash memory, they remap the write requests to empty areas of flash memory while maintaining the mapping information. They also propose the cost-benefit policy based on a value-driven heuristic function as block-recycling policy.

For native flash file system, JFFS (Journaling Flash File System) [6] and YAFFS (Yet Another Flash Filing System) [7] have been developed. Both file systems are based on LFS (log-structured file system) architecture [8]. In LFS, all file system modifications are logged sequentially. This method is well harmonic with flash memory, because the flash memory is hard to update due to its unique feature. However, it is not easy to use the file systems for mobile devices, because the logging architecture requires a large amount of memory and long mounting time.

J.S. Kim et al. propose more practical FTL algorithm called Log-block FTL algorithm for compact flash [9]. Log-block FTL algorithm combines the two different granularities in address translation. It is motivated by the idea that coarse-grain address translation reduces resources required to maintain translation information, which is crucial in mobile consumer products in point of cost and power consumption, whereas fine-grain address translation is efficient in handling small size writes. Because Log-block FTL is quite competitive, our real-time FAT file system proposed in this paper is based on it.

FAT file system with FTL has been widely used as the standard file system for commercial mobile devices. USB disks and memory cards use FTL and they are formatted with the FAT file system for compatibility. The FAT file system has been originally developed for MS-DOS system. It is well known, simple, and fast mountable, but it is not adequate for mobile devices because its robustness is not enough for sudden power failures. Therefore, there have been various trials to enhance the robustness of FAT file system.

M.S. Kwon et al. propose KFAT file system [10] which is compatible with the original FAT file system and also guarantees the integrity of the FAT table and directory entries. Microsoft proposes TFAT [11] for the same purpose. They use duplicated twin FAT tables to protect the file system from the sudden system crash. AtomicWriteFTL [12] is another approach to enhance the robustness of the FAT file systems using FTL. It supports atomic write function of multiple sectors to avoid inconsistency among distributed file system metadata. It is based on Log-block FTL algorithm, but it enhances the robustness of the file system fundamentally without much cost.

L.P. Chang et al. propose a real-time garbage-collection mechanism, which provides a guaranteed performance for hard real-time systems [13]. This mechanism supports non-real-time tasks also so that the potential bandwidth of the storage system can be fully utilized. A wear-leveling method, which is executed as non-real-time service, is presented to resolve the endurance problem of flash memory. However, it does not seem to be a proper solution for mobile multimedia devices.

Many previous researches on multimedia file system are based on hard disk rather than flash memory [14][15]. Their major issue is how to handle multiple read requests from multiple processes in a streaming server. Multimedia file system based on flash memory has a quite different issue. The read speed of flash memory is much faster than the write speed, and a mobile device is mainly used by a single user. Therefore,

a single stream and a single thread are enough for mobile multimedia flash file system, and the issue is how to record single multimedia stream stably by single thread.

III. BACKGROUND

A. Characteristics of NAND Flash Memory

There are two types of flash memories: NOR flash memory and NAND flash memory. They have similar in function, but they have different interfaces and internal architectures. Since NAND flash memory is widely used in the mobile multimedia devices, we mainly focus on NAND flash memory rather than NOR flash memory in this paper.

NAND flash memory has a unified bus for both address and data lines. Therefore, through the common bus, address and data are transmitted alternatively by the predefined sequence. Internally, NAND flash memory has serial cell architecture and it can be accessed only by page unit.

NAND flash memory consists of multiple blocks, and each block consists of multiple pages. The number of blocks in a chip is dependent to the capacity of the device, and the number of pages in each block is fixed in the same device. Each page consists of main arrays and spare arrays. Main array is used to store user data, and spare array to store additional useful information such as ECC (error correction code) for the main array, a mark for bad block, logical sector number, and so on.

There are two types of NAND flash memories. One is a small block NAND flash memory. In this memory, each page consists of 512 bytes main array and 16 bytes spare array and the number of pages in each block is 32. The other is a large block NAND flash memory. In this memory, each block has of 64 pages, and each page is composed of 2,048 bytes main array and 64 bytes spare array.

Fig. 2 shows the overall structure of a small block NAND flash memory. Each block has 32 pages and each page consists of 512 bytes main array and 16 bytes spare array. The total size of each page is 528 bytes and that of each block is (528 * 32) bytes.

Note that the page size excluding spare array is 512 bytes and it is the same as the size of each sector in the file system. In this paper, a sector implies the group of data contents to write and a page implies the place in the block. However, we will sometimes use two terms interchangeably.

In NAND flash memory, *write operation* means the process of changing data bits from 1 to 0. In other words, write operation is to discharge the corresponding transistor cells. Once a bit is changed from 1 to 0, it can not be restored again by additional writes, because charge can not be refilled by write operation. Therefore, another process is required to charge data bits. Such the special charge process is called *erase operation*. Since charging requires some time to process, each erase operation takes a relatively very long time, compared to read and write operations. Moreover, for efficiency, the erase operation takes place in block unit, not in page unit. The mismatch between write operation unit and erase operation unit is the main obstacle for developers.

To update a page of flash memory, the whole block which includes the updating page may be required to be erased. Because the block may contain other valid pages, a special method is required not to lose the data of the block and not to degenerate efficiency. For this purpose, FTL is developed, as described in Section I.

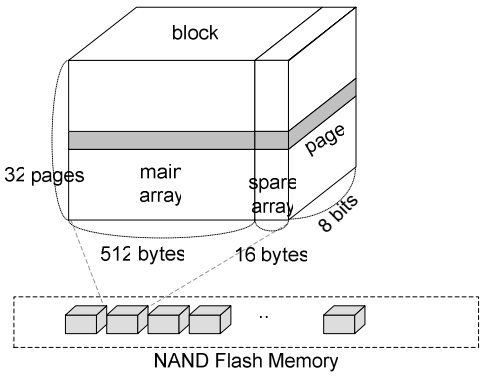

Fig. 2. The overall structure of a small block NAND flash memory.

B. Log-block FTL algorithm

Log-block FTL algorithm [9] has been proposed by Jesung Kim, et al. for compact flash system with restricted resources. In Log-block FTL algorithm, there exist two types of blocks: a log block or a data block. They have different page allocation policies, respectively. A log block uses so called *out-of-place* policy, whereas a data block uses so called *in-place* policy. In-place policy means that it allocates a new page to a sector so that the position of the page itself may imply the logical sector number. In contrast, out-of-place policy means that a sector is allocated to the next available page in the corresponding block, regardless of its logical sector number.

Fig. 3 describes how these two page allocation policies work. In this figure, it is assumed that a block consists of 4 pages to help understanding. The shadowed boxes in this figure denote that the pages are occupied, and the numbers inside the boxes represent their page numbers, respectively. In case of the in-place policy as shown in Fig. 3(a), no mapping information is needed because the position itself represents its sector or page number in that block. However, the updating process of a page/sector is not easy because flash memory can not be updated directly in page unit without block erasure. In case of the out-of-place policy as shown in Fig. 3(b), new or updated sectors are sequentially appended in a block. This policy is more efficient for updating a page/sector than the in-place policy, but additional mapping information is required to indicate where the logical sector is written.

In Log-block FTL algorithm, all new or updated pages are written to the corresponding log block, not to data blocks. It seems efficient in sense that it writes only updated pages without erasing data blocks. Since the out-of-place policy is applied to log blocks, several pages with different contents for the same logical page may exist in the same log block. The page with the recent update can be searched in the

corresponding log block, if it is updated. Otherwise, it will be easily retrieved from the corresponding data block.

However, when a log block becomes full, a merge process is forced to be executed. The full log block is merged with the corresponding data block to a new data block. It takes relatively long time and it is one of the reasons why the FAT file system with Log-block FTL produces irregular write responses.

Fig. 4 shows an example of the merge process in Log-block FTL algorithm. Initially, there is no log block assigned to data blocks as shown in Fig. 4(a). In Fig. 4(b), a log block is created and assigned to the second data block to write sector 6, 6 and 5. We can calculate the logical block number from the logical sector number. For example, since there are 4 pages in each block, sectors 5 and 6 are to be allocated into the second block. Another sector write makes the log block full, and then the log block and the old data block (the second one) are merged to a new data block as shown in Fig. 4(c). After the merge process, the log block and the old data block become free blocks. Log-block FTL maintains block-mapping information in map blocks to switch the blocks effectively.

C. *FAT File System*

The FAT (File Allocation Table) file system was originally developed as a simple file system suitable for floppy disk drives whose size is less than 500K Bytes. Then, it has been enhanced to support the larger and larger media. Currently there are three categories: FAT12, FAT16 and FAT32. The basic criterion is the bit size of each entry in the FAT data structure on the disk. Note that the size means how many sectors the system can address and manage. In other words, it means how large disk it can support.

Fig. 5 shows the structure of the conventional FAT file system. It consists of four major parts: BPB (Boot Parameter Block), FAT1 / FAT2, root directory, and cluster region. BPB is a data structure at the first position of the disk. It includes the boot sector, file system information, and additional program code. The boot sector of BPB is used for booting process by BIOS of PC. In addition, BPB includes the configuration parameters of the FAT file system such as the size of a cluster and the number of FAT table entries. This information is necessary to mount and access a volume which is formatted by FAT file system.

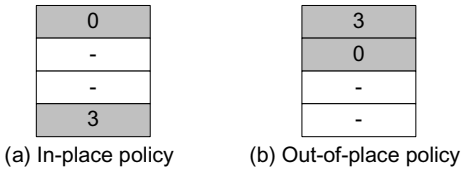

Fig. 3. Page allocation policies: (a) In-place police (b) Out-of-place policy.

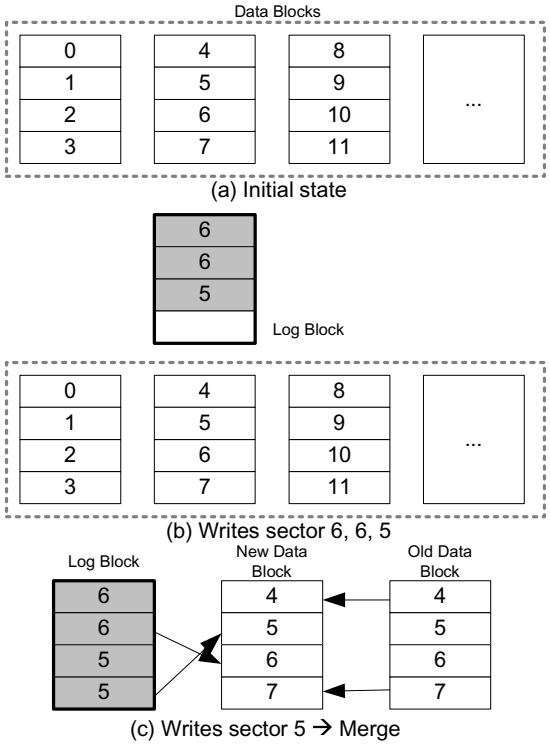

Fig. 4. Log-block FTL algorithm and merge process: (a) The initial state. (b) Sectors 6, 6 and 5 are written. (c) Additional write of sector 5 invokes a merge process.

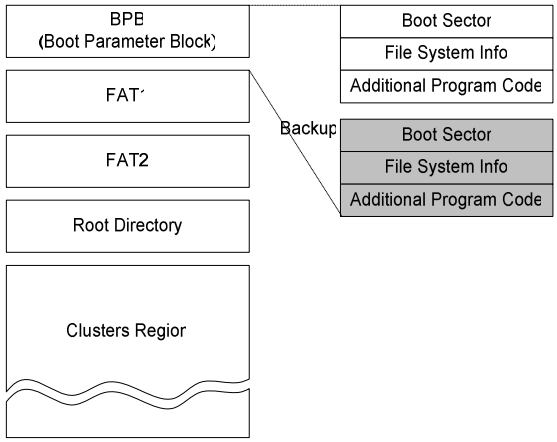

Fig. 5. The structure of FAT file system: BPB, FAT1, FAT2, Root directory, and Cluster region.

FAT1 and FAT2 are the core part of the FAT file system. A file of FAT file system has a directory entry and occupies multiple data clusters. FAT table is used to interconnect the directory entry and the multiple data clusters of each file. Directory entry of FAT file system contains the file name, the file size, the time information, and the starting cluster number. With the cluster number in FAT table, the next cluster number can be calculated, because the data clusters are linked in the FAT table. If the value of FAT table entry is 0xffff (meaning EOF), it means the cluster is the last cluster in a file. Otherwise, the value means the number of the next cluster in the file. For robustness, the FAT file system maintains duplicated FAT tables. Normally, two FAT tables are called FAT1 and FAT2.

Root directory area of the FAT file system includes directory or file entries of root directory. Clusters region is the largest area of the FAT file system volume, and it is used for file data clusters and the directory entries except root directory.

IV. Techniques for Real-Time FAT File System

A. Sector Reservation method

Due to the characteristics of flash memory, a page update may require some additional operations such as the block erasure to be performed implicitly. Therefore, when a page is to be updated, FTL usually allocates a new page, and writes the updated contents to the new page, and invalidates the old one, instead of directly updating the page.

As mentioned earlier in the previous section, the Log-block FTL algorithm writes all new or updated sectors to the corresponding log block, not directly to data blocks. Since all the new or updated sectors are appended to the corresponding log block, the log block can become full soon, requiring the merge process. During the merge process, the full log block is combined with the corresponding data block and a new data block is constructed. Since the process takes long time and moreover it occurs irregularly, it causes the FAT file system with Log-block FTL to produce irregular write responses.

Note that our goal in this paper is to record a huge multimedia stream stably. Therefore, such the process should be avoided during recording the multimedia stream. To achieve this goal, we have developed a new technique called *Sector Reservation method*.

In this method, a sector is written to a data block directly if the target page of the corresponding data block is empty.

Fig. 6 shows how the sectors are written in this method. Before we write a new sector to its corresponding log block, we first read the spare array of its target page in the corresponding data block to check whether it is empty or not. If all sixteen bytes of the spare array are 0xff, the page is assumed to be empty. If the page is not empty, the sector is written to its corresponding log block as in the original Log-block FTL.

Fig. 7 shows the flowchart of the sector writing algorithm of our method. Shadowed parts are added compared to the original method.

In order to improve the performance of our new sector writing algorithm, as many target pages of the data block should be empty as possible. Therefore, in our method, sectors are reserved as soon as possible when they are invalidated, making the corresponding pages clean or empty.

Our method looks very similar to the normal sector deletion function in the original Log-block FTL. But, they are not the same. Sector deletion is used to enhance FTL performance by indicating which sectors are invalidated. Contrary to the sector deletion, our method is used to maintain the write responses deterministic.

Our sector reservation function receives the range of pages to reserve as argument. If there exists a log block assigned to the target data block, sector reservation function constructs a new data block from a free block, the old data block, and the log block. It is very similar to the merge operation except the pages to reserve are remained empty during the operation.

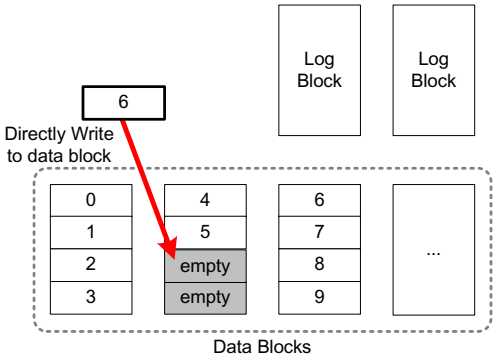

Fig. 6. An example of sector writing operation in Sector Reservation method: When the target page of the corresponding data block is empty.

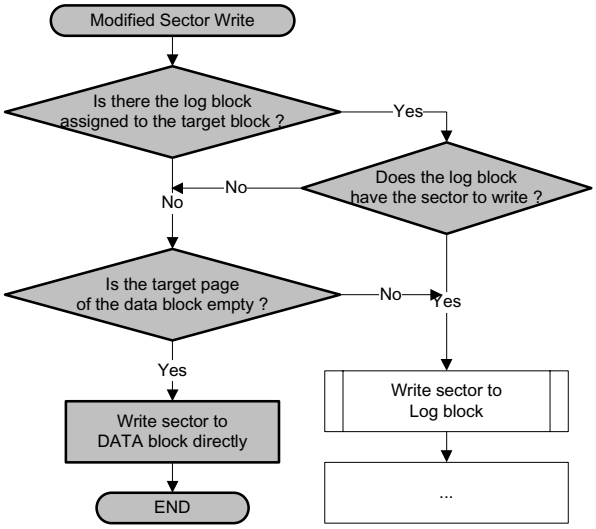

Fig. 7. The flowchart of the modified sector writing algorithm in the Sector Reservation method.

Fig. 8 and 9 show how Sector Reservation method works when sectors 4 and 5 are to be reserved. Fig. 8 shows an example in case that there exists the corresponding log block. A new data block is constructed by copying the pages of the old data block and the log block except for the sectors to reserve. Fig. 9 shows an example in case that no log block is assigned to the data block. Since there is no log block assigned to the data block, the process is simpler.

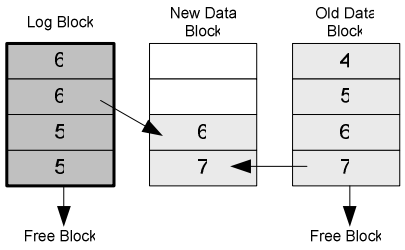

Fig. 8. An example of sector reservation when there exists the corresponding log block: Sectors 4 and 5 are to be reserved.

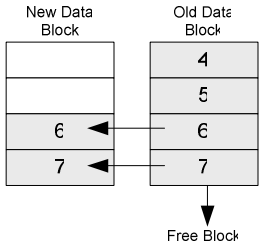

Fig. 9. An example of sector reservation when no log block is assigned to the data block: Sectors 4 and 5 are to be reserved.

B. *All Cluster Pre-Allocation (ACPA) method*

In the existing FAT file system for flash memory, FAT table contains cluster allocation information for each file and thus it should be updated whenever a cluster is allocated for a file. In case that the size of a file is not fixed as in case of real time voice recorded data, additional clusters are allocated to the file one by one, and thus related FAT table should be frequently updated accordingly by the file system. Normally, this process takes a relatively long time in flash memory and degrades the performance of the file system, while making the response time irregular. In desktop computer environment, buffer cache can be used to avoid this kind of performance degradation, but it is not recommended for mobile devices because file system data may be lost by sudden power failure.

In this paper, we devise a new method called ACPA (All Clusters Pre-Allocation) to reduce the number of FAT modifications. In this method, all the available clusters are pre-allocated at once for a file when the file is created. Thus the file can be written without further cluster allocation which requires FAT update. Instead, when the file is closed, unused free clusters are released. Such method avoids frequent FAT modification even in case of large multimedia files.

ACPA uses two parameters in each directory entry of the FAT file system: *DIR_FileSize* representing the file size in byte unit and *DIR_FstClus* the starting cluster number.

Fig. 10 shows a simple example of cluster allocation and release in ACPA method. Fig. 10(a) assumes that five free clusters are left in a volume and the size of each cluster is 2,048 bytes. When a multimedia file is created, all free clusters (from cluster 2 to cluster 6) are allocated at once to the multimedia file as shown in Fig. 10(b). At this time, the size of the created file is zero byte, but 5 clusters are allocated to the file. In this allocation process, the corresponding 5 entries of FAT table are modified. Allocated in advance but unused clusters will be released when the file is closed. In this example, it is assumed that the file is closed after 5,000 bytes are written. Because we assume the size of cluster is 2,048 bytes, the file needs only three clusters out of 5 pre-allocated clusters, releasing clusters 5 and 6. Fig. 10(c) shows the status after the file is closed.

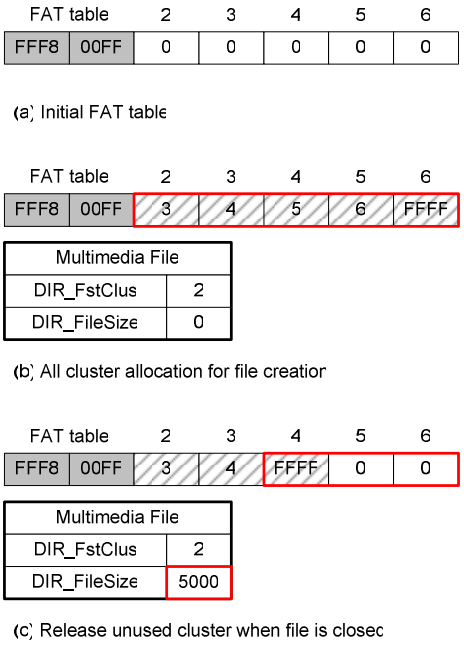

Fig. 10. Cluster pre-allocation and release of ACPA method: (a) Initial state. There are five free clusters. (b) When a multimedia file is created. All the remaining clusters are allocated to the multimedia file and thus five entries of FAT file system are modified. (c) When the file is closed. Two Unused clusters are released.

Note that this method is also helpful for safe recovery, because user data is never lost even in case of sudden power failure. If the system is crashed before the multimedia file is closed, there will be no free cluster when the volume is mounted again. This exceptional case, however, can be easily managed if we can locate the file to which all the free clusters were allocated.

The performance effect of ACPA method can be better or worse than original cluster allocation method. It is dependent on the size of free spaces in FAT file system. Since there are usually too many free clusters in a volume, file creation and file closing processes will take a long time. Therefore, we may need to modify ACPA and limit properly the range of pre-allocation for better performance. This topic will be addressed in our future work.

C. *FAT File System and Sector Reservation method*

Our Real-Time FAT file system adapts Sector Reservation method. When a file is deleted, the pages/sectors which were allocated to the deleted file are reserved automatically. If there are only sequential writes, it guarantees that a new data cluster is previously reserved by Sector Reservation method. Therefore, the responses of sequential writes in our Real-Time FAT file system can be deterministic and uniform. However, the file deletion time may be prolonged by Sector Reservation method.

D. *Deterministic write responses*

Our Real-Time FAT file system guarantees that the sequential write requests of (S_{main} * N) bytes are completed by N spare array read operations and N page-write operations,

where S_{main} denotes the size of main array of each page in NAND flash memory. For a small block NAND flash memory, S_{main} is 512 bytes. For example, our Real-Time FAT file system guarantees that 32K bytes write is completed by 64 reads on the spare array and 64 page-writes in a small block NAND flash memory. There also may be some CPU computational overhead, but it is ignorable.

The spare array read time, T_{read_spare} can be represented as in (1), where T_R is the time for page-read operation from the cell to the register of NAND flash memory, S_{spare} is the number of bytes of each spare array, and T_{tr} is byte transfer time from NAND flash memory to CPU.

$$T_{read\_spare} = T_R + (S_{spare} \times T_{tr}) \quad (1)$$

The page write time, T_{wirte_page} can be expressed as in (2), where S_{page} is the number of bytes of each page, T_{tr} is the byte transfer time from CPU to NAND flash memory, and T_{PROG} is the page programming time.

$$T_{write\_page} = (S_{page} \times T_{tr}) + T_{PROG} \quad (2)$$

Therefore, the total time for writing (S_{main} * N) bytes can be expressed as in (3) for our Real-Time FAT file system. Note that T_{tr} is dependent on the bank configuration of host processor.

$$T_{(S_{main} \times N)_{write}} = N \times [\{T_R + (S_{spare} \times T_{tr})\} + \{(S_{page} \times T_{tr}) + T_{PROG}\}] \quad (3)$$

For example, (3) can be reformed as in (4) for the small bock NAND flash memory described in [16].

$$T_{(512 \times N)_{write}} = N \times [\{10\mu s + (16 \times T_{tr})\} + \{(528 \times T_{tr}) + 200\mu s\}] \quad (4)$$

V. EXPERIMENTAL RESULTS

We modified the existing FAT file system using the original Log-block FTL and implemented a prototype Real-Time FAT file system with our new techniques. The system is running on ARM9 board with a 32M bytes NAND flash memory and 203 MHz CPU. We also developed an emulation tool and applied it to measure the response times of the existing FAT file system as well as our new system.

First, we simulated the case when a very large multimedia file was created. In this experiment, we configured the cluster size of FAT file system as 2K bytes (4 pages or sectors in this case), and then generated more than 64 sequential write requests whose sizes are 2K bytes, respectively. After that, we measured and compared the response time of each write request for the following four different systems: (1) the existing FAT file system with the original Log-block FTL, (2) the modified FAT file system using ACPA method, (3) the modified FAT file system using Sector Reservation method, and (4) our Real-Time FAT file system, that is, the modified FAT file system with both ACPA and Sector Reservation methods.

Fig. 11 shows the response times for the existing FAT file system with the original Log-block FTL. Each write request took from 2.4ms up to 37ms. The worst case took more than fifteen times longer than the best case. It is noted that two kinds of periodical peaks are found. The small peak occurs for every 8 write requests, and the high peak for every 32 requests. In other words, the small peak occurs after writing every 16K bytes, and the high peak occurs after writing every 64K bytes.

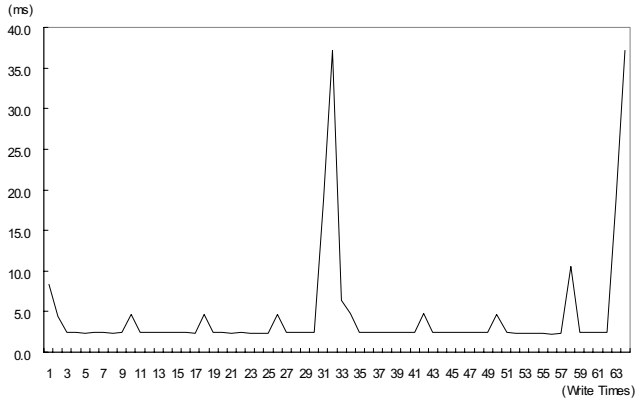

Fig. 11. Write response times of the existing FAT file system with the Log-block FTL.

Our analysis shows that the small peaks are caused by merge processes occurred in FTL. In FTL, sequential 8 write requests of 2K bytes are translated to 32 page-writes. It means that a log block becomes full after those operations. In that case, the log block is converted to data block by so called *switch merge* process in FTL. Such the switch merge process includes one block erasure operation and one page write operation. Actually, we can find another peak which occurred at the 58th write request. It was because some additional updates on the map block occurred for the switch merge.

Our analysis also shows that the high peaks are caused by frequent modifications on FAT table entries. Because the cluster size is 2K bytes, every write request requires another cluster allocation in the existing FAT file system. It means that FAT table entry should be modified each time, creating a page-write operation. In the Log-block FTL, the log block assigned for FAT table becomes full for every 32 FAT modifications, because one block consists of 32 pages. In this case, a merge process occurs in the Log-block FTL. Because each FAT modification creates a new page in the log block with out-of-place policy, a merge process, which takes more time than the switch merge, occurs. Moreover, such the process is executed twice, since there are two FAT tables (FAT1, FAT2) in the FAT file system.

Fig. 12, 13, and 14 show the response times for the modified FAT file system with different options, respectively. We depict the figures separately to demonstrate the effect of each of our new techniques proposed in this paper.

Fig. 12 shows the result for the modified FAT file system with ACPA only. As expected, all the high peaks disappeared. Moreover, the best write response time reduced from 2.4ms to 1.3ms and the worst case time reduced from 37ms to 3.6ms dramatically. This implies that ACPA is helpful to avoid frequent FAT modifications.

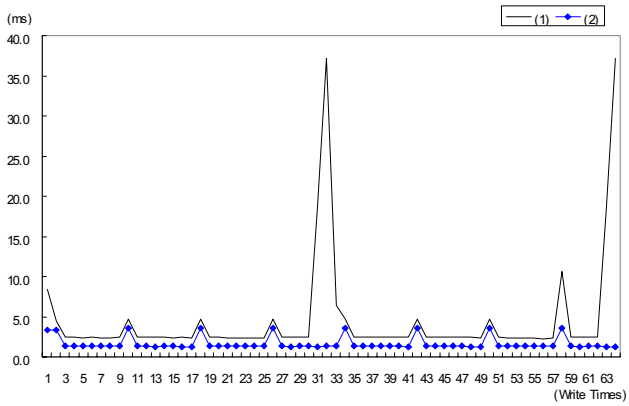

Fig. 12. Write responses before and after ACPA method is applied: (1) the existing FAT file system with the Log-block FTL, (2) the modified FAT file system with ACPA method.

Fig. 13 shows the result for the modified FAT file system with Sector Reservation method. It is notable that all the small peaks disappeared. It is because our Sector Reservation method effectively reduces the chance that the log block becomes full and helps to avoid the occurrence of the merge process in advance. The best response time slightly increased by 0.1ms due to spare array read operations for sector reservation, while the worst was unchanged.

Fig. 14 shows the experimental results for our Real-Time FAT file system where both ACPA method and Sector Reservation method are applied together. All write response times were almost the same. The average response time is 1.4ms and the standard deviation is 0.016ms. Note that all the peaks in Fig. 11 disappeared when both of our new techniques were applied together, making the write responses uniform.

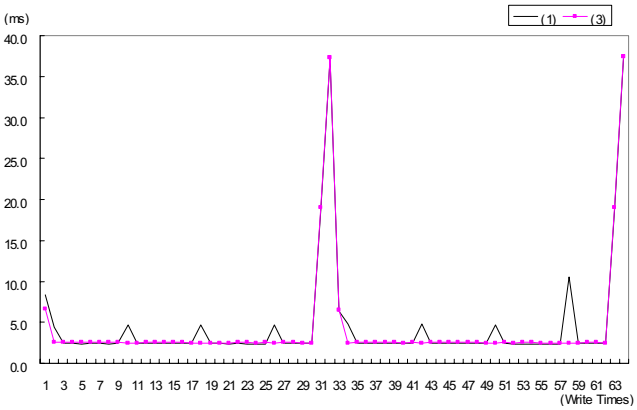

Fig. 13. Write responses before and after Sector Reservation method is applied. (1) The existing FAT file system with the Log-block FTL, (3) The modified FAT file system with Sector Reservation method.

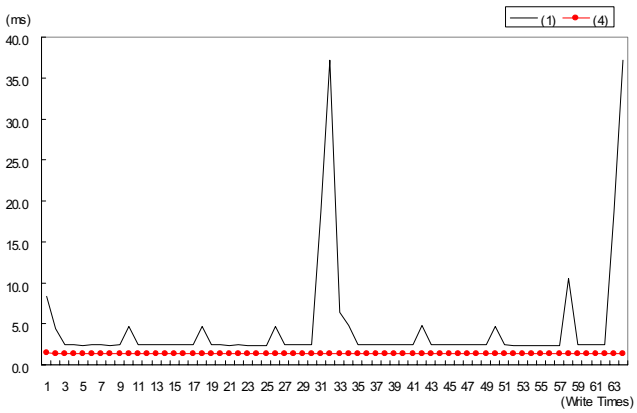

Fig. 14. Write responses before and after both ACPA and Sector Reservation methods are applied. (1) The existing FAT file system with the Log-block FTL, (4) Our Real-Time FAT file system.

As another experiment, we tested the case when the data size of each sequential write request was getting changed. We measured the response times for (512 * N) bytes write requests, where N was changed from 1 to 32.

Fig. 15 shows the write response times for the four systems. We can recognize that the responses are very irregular for the first three cases, but not for the last one. The response times for the last one increase linearly in proportion to the size of each write request. In other words, the write responses for our new system are very deterministic. With this result, we can estimate the time for one sector write in our Real-Time FAT file system, and the response time for each write request of (512 * N) bytes can be represented as in (5).

$$t_{(512\times N)_{write}} = N \times 352\mu s \tag{5}$$

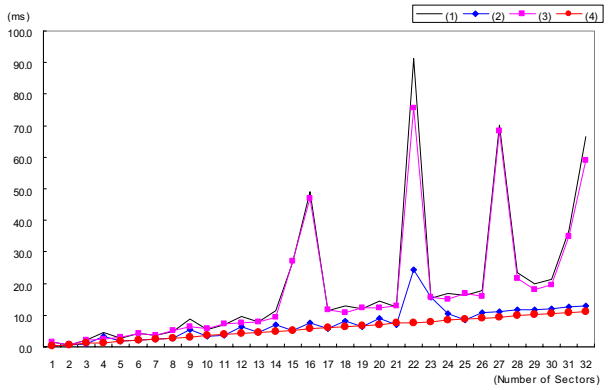

Fig. 15. Write responses for (512 * N) bytes: (1) the old FAT file system with the Log-block FTL, (2) the FAT file system modified with ACPA, (3) the modified FAT file system with Sector Reservation method, (4) our Real-Time FAT file system.

VI. CONCLUSION

In this paper, we have considered the problems of the existing FAT file system with the Log-block FTL, and then developed a new Real-Time FAT file system with two novel techniques called ACPA method and Sector Reservation

method to solve the problems. Our experimental results show that these two methods work very effectively and the response times of our new system are quite deterministic and uniform. It implies that our new file system enables the mobile multimedia devices to record multimedia stream stably.

However, the performance of our new techniques needs further investigation. First, we have experimented only the cases of the sequential write requests because our main goal in this paper is to record a huge multimedia stream stably. We need consider more general and sophisticated cases for the commercial mobile products. In addition, since there are so many free clusters in a volume, file opening and/or closing processes in ACPA may take too much time. We can modify ACPA and limit properly the range of pre-allocation for better performance. These topics will be addressed in future work.

Currently, we have considered the problems of the FAT file system with the original Log-block FTL. We also need further research on different file systems as well as different FTL's.

REFERENCES

[1] Microsoft, "FAT File System: The Story Behind the Innovation," http://www.microsoft.com/mscorp/ip/tech/fathist.asp, 2003.

[2] Microsoft, "FAT32 File System Specification," http://www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx, 2000.

[3] Intel Corporation, "Understanding the flash translation layer (FTL) specification," http://developer.intel.com.

[4] M. Wu, and W. Zwaenepoel, "eNVy: a non-volatile, main memory storage system," in *Proceedings of the 6th International Conference of Architectural Support for Programming Languages and Operating Systems*, 1994, pp.86-97.

[5] A. Kawaguchi, S. Nishioka, and H. Motoda, "Flash-Memory Based File System," in *Proceedings of '95 Winter USENIX Technical Conference*, 1995, pp.155-164.

[6] D. Woodhouse, "JFFS: The Journaling Flash File System," in *Ottawa Linux Symposium*, 2001.

[7] Aleph One Company, "Yet Another Flash Filing System," http://www.aleph1.co.uk/yaffs.

[8] M. Rosenblum, and J. K. Ousterhout, "The design and implementation of a log-structured file system," *ACM Transactions on Computer Systems (TOCS), Vol. 10, Issue 1*, 1992, pp.26–52.

[9] J.S. Kim, J.M. Kim, S. H. Noh, S. L. Min, and Y.K. Cho, "A Space-Efficient Flash Translation Layer for Copactflash System," in *IEEE Transactions on Consumer Electronics*, 2002, pp.366-375.

[10] M.S. Kwon, S.H. Bae, S.S. Jung, D.Y. Seo, and C.K. Kim, "KFAT: Log-based Transactional FAT Filesystem for Embedded Mobile Systems", in *Proceedings of 2005 US-Korea Conference, ICTS-142*, 2005.

[11] Microsoft, " Transaction-Safe FAT File System", http://msdn.microsoft.co.m/library/default.asp.

[12] S.H. Park, J.H. Yu, and S. Y. Ohm, "Atomic Write FTL for robust flash file system," in *Proceedings of the Ninth International Symposium*, 2005, pp.155–160.

[13] L.P. Chang, T.W. Kuo, and S.W. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," in *ACM Transactions on Embedded Computing Systems (TECS)*, Vol. 3 Issue 4, 2004, pp. 837 – 863.

[14] P. V. Rangan and H. M. Vin, "Designing file systems for digital video and audio," in *Proceedings of the thirteenth ACM symposium on Operating systems principles SOSP '91*, Vol. 25, Issue 5, 1991, pp.81–94.

[15] B.S. Ahn, S.H. Sohn, C.Y. Kim, G.I. Cha, Y.C. Baek, S. I. Jung, and M.J. Kim, "Implementation and evaluation of EXT3NS multimedia file system," in *Proceedings of the 12th annual ACM international conference on Multimedia*, 2004, pp.588-595.

[16] Samsung Electronics, "K9K1G08R0B 128M x 8 Bit NAND Flash Memory Data sheet," www.samsung.com.

Sunhwa PARK received BS and MS degrees in computer science, both from Seoul Women's University, Korea, in 1998 and 2000. She is PhD candidate of Seoul Women's University.

Her research interests include flash memory software, hardware software co-design, and general embedded systems.

Seong-Yong OHM received the B.E., M.S., and Ph.D. degrees in Computer Engineering from Seoul National University, Seoul, Korea in 1985, 1987, and 1992 respectively. Since March 1996, he has been working with Department of Computer Science and Engineering at Seoul Women's University, Seoul, Korea as Professor. His main interests include high-level synthesis, hardware software co-design, mobile system, computer graphics, and embedded systems.