This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE CCNC 2006 proceedings.

MNFS: Mobile Multimedia File System for NAND Flash based Storage Device

Hyojun Kim
Samsung Electronics Co., Ltd.,
Korea 442-600
zartoven@samsung.com

Youjip Won
ECE division of Hanyang University
yjwon@ece.hanyang.ac.kr

Abstract

In this work, we present a novel mobile multimedia file system, MNFS, which is specifically designed for NAND flash memory. It is designed for mobile multimedia devices such as MP3 player, Personal Media Player (PMP), digital camcorder, etc. Our file system has three novel features important in mobile multimedia applications: (1) predictable and uniform write latency, (2) quick file system mount, and (3) small memory footprint. We implement the proto-type file system on ARM9 embedded platform. In experiments, MNFS exhibits uniform I/O latency for sequential write operation. It is mountable within 0.2 seconds, and available with only 34Kbytes heap memory for 128Mbytes volume. Compared to YAFFS which is the de facto standard file system for NAND flash memory, the mounting time is 30 times faster and the heap memory usage is only 5% of YAFFS usage.

Keywords – Multimedia, Flash Memory, File System, I/O latency, Mobile Device

1. Introduction

1.1 Motivation

Due to the rapid advancement in computer architecture, storage system and etc., we can now bring large amount of data in daily life in extremely portable manner and further can access it instantaneously. Mobile multimedia devices, e.g. MP3 player, Portable Multimedia Player, digital camcorder are typical examples. Current state of art file system, storage system, and encoding/decoding technology opens up an opportunity for the end users to enjoy various multimedia services in extremely portable fashion. Different from the workloads which can be observed in commodity computer system, the I/O workload generated in mobile multimedia device exhibits rather unique characteristics. In MP3 player, for example, large size file which is usually more than 2-3 Mbytes is first cached onto memory and is played back. This is primarily to reduce the energy consumption and to make the storage device in inactive state whenever it is possible. In digital camcorder, the captured images are continuously feed into storage device and file system is required to handle isochronous write requests in efficient fashion. That is, multimedia files are sequentially accessed and not updated often. Meanwhile, one of the main drawbacks of NAND flash device is that a page is not updatable without block erasure. Because multimedia files are rarely updated, NAND flash based storage subsystem can provide very efficient solution for mobile multimedia device. However, there are a number of issues we like to address in this work. While the NAND flash device carries very desirable characteristics for mobile multimedia application, the commodity file system leaves much to be desired to exploit the workload characteristics and the characteristics of storage media itself.

Irregular write-responses are major problem of existing file system for mobile devices. To record multimedia data generated in real time, predictable and uniform write latency is required. It will be more critical as the bandwidth of mobile multimedia is increasing. Quick mounting is basic requirements for mobile devices. Nobody wants to wait tens of seconds to take picture with digital camera. Small memory footprint is also important for mobile devices. Generally, available memory is restricted in mobile devices because of cost and other reasons. Of course, the performance of file system is important.

1.2 Related works

To use flash memory as file system storage, Flash Translation Layer (FTL) was developed [1]-[3]. FTL is a software layer which emulates hard disk with flash

memory. FTL covers flash memory's write-once, bulk erasable characteristics with its mapping algorithm.

Flash native file systems were designed only for flash memory to remove FTL overhead. Microsoft Flash File System is a flash native file system using linked-list data structure for NOR flash memory [4]. Later, Log-structured File System (LFS) structure [5] was adopted to flash file system. JFFS is the first flash file system that adopts LFS structure based on NOR flash memory. Later, it was developed to JFFS2 and widely used in Linux domain [6]. JFFS2 was then later extended to support NAND flash memory, but it could not show optimized performance because NAND flash memory has different characteristics. In 2000, YAFFS was designed for NAND flash memory [7]. It is another log-structured flash file system which is optimized for NAND flash memory.

Currently, LFS style flash file system has become major trend. Despite its widespread adoption, it suffers from important problems. It requires large amount of memory for mapping table. Further, file system mount latency is very large. Unless these issues are resolved properly, it cannot be adopted easily in mobile device.

To reduce mapping table size, Li-Pin Chang and Tei-Wei Kuo proposed a flexible management scheme for large-scale flash-memory storage systems [8]. However, to apply this method on real FTL or flash file system, the number of consecutive sectors to be written must be known when the first sector is written. POSIX compliant file system interface does not carry this information and therefore to realize this feature we need to re-define the file system interface.

Garbage collection of log-structured flash file system and sector mapping FTL is the prime cause for largely variable write latency. In garbage collection process, obsolete logs are cleaned by copying some pages and erasing block. It takes a long time compared to single page write. Therefore a new garbage collection method was proposed by Li-Pin Chang et al. for hard real-time applications [9]. They proposed to use background tasks which are activated periodically for background garbage collection to guarantee deterministic write-responses of real-time tasks. This scheme requires real-time operating system.

In this paper, we present MNFS, a multimedia file system for NAND flash based storage device focusing on workloads of mobile multimedia files. We assume that mobile multimedia files have larger size and rarely updated. The goal was to optimize sequential writing performance rather than overwriting performance. The design criteria called for the MNFS to be deterministic and have uniform write-responses.

We implemented prototype MNFS on ARM9 platform. For evaluation, we examined the performance of MNFS with YAFFS [7] and FAT file system [11]. It is observed that MNFS successfully address the issue in existing flash file system and exhibits much better write performance with less variable latency.

The rest of this paper is organized as follows. Next section explains key idea of MNFS design. Following section describes experimental results. Conclusions and future work are described in final section.

2. MNFS Design

Key ideas of MNFS are (1) hybrid mapping, (2) block based file allocation, (3) iBAT (in-core only Block Allocation Table) and (4) upward directory representation. By these methods, MNFS achieves uniform write-responses, quick mounting, and small memory footprint.

2.1 Hybrid mapping algorithm

In NAND flash memory device, smallest unit of write operation is a page (512-2,048 Bytes). Due to the physical characteristics, write operation requires that the existing contents are first erased. The unit of erase operation is a block and a block consists of 32 or 64 pages. Therefore, even though the write size is much smaller than single block, irrelevant pages in the same block needs to be erased. There are a number of different approaches to resolve this issue. SmartCard™ [10] uses block mapping algorithm. To update page, it allocates a new block and copies original pages to the block except the page to be updated. The new page is written to the new block which in turn causes the block mapping information to change. YAFFS [7] uses page mapping algorithm. It writes a log to any empty page, and maintains the page-level mapping table. Page mapping method is efficient for frequent updates, but requires large translation table in main memory. Meanwhile block mapping method requires small translation table, but not efficient for frequent updates.

In MNFS, we propose hybrid mapping algorithm. We exploit the access characteristics of the respective region and apply block and page mapping algorithm based upon the frequency of update operation. File system metadata is updated whenever a file is created, deleted, and appended. Therefore, it is to be updated frequently and MNFS uses page mapping algorithm (log-structured method) for file system metadata. On the other hand block mapping algorithm is used for user data because it is rarely updated in mobile multimedia devices. Figure 1 shows basic structure of MNFS using hybrid mapping algorithm.

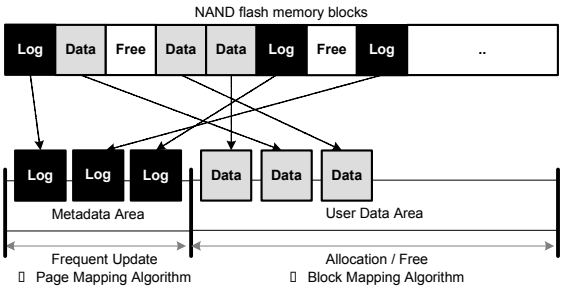

Figure 1 File system organization of MNFS

MNFS uses log structure to manage file system metadata. The log blocks are sorted in temporal order and managed by linked list. The first page of a log block contains log block header in main area, and log block sequence number in spare area. Log block header contains file system configuration information, e.g. partition information and MNFS signature. The remaining pages of log block are used for directory entries. Each directory entry represents a file or a directory. It contains file (or directory) name, size, attribute, creation date and time, etc. A unique ID is assigned to each directory entry, and stored in spare area. Because directory information resides in log block, updating of the directory entry is done by writing new directory entry to the end of the log. There can be multiple entries which have same ID, at that time the most recently written entry is valid. Figure 2 shows the structure of the log block.

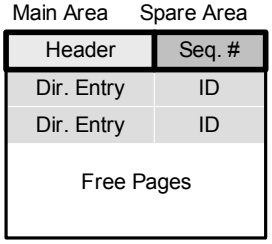

Figure 2 Log block structure of MNFS

2.2 Block based file allocation

Multimedia file, e.g. music or video clip, is order of magnitude larger than the text based file. Therefore MNFS uses larger allocation unit than the block size (usually 4 Kbytes) of legacy general purpose file system. MNFS defines the allocation block size of file system as the block of NAND flash memory. The block size of NAND flash memory ranges from 16Kbyte to 128Kbyte and this size is device specific. There are important advantages of defining device block as file allocation unit.

Because erase operation in NAND flash memory is performed in block unit, all allocated blocks to a file can be erased when the file is deleted. NAND flash memory must be erased to update by its nature, and MNFS processes the erase job on file deletion time. Other flash file systems, such as JFFS2 and YAFFS, do not erase NAND flash memory when a file is deleted, and later, NAND flash memory is erased by garbage collector during writing process. Because the block erasing time is about 6 – 10 times longer than page writing time, erase operation during write request makes it is hard to get uniform write-responses. Because MNFS erases all block when the blocks are released, MNFS can write pages in constant time without block erasing. Of course, file deletion time of MNFS becomes longer than other file systems. But, we think, uniform write-responses are much more important than fast file deletion. And, the file deletion time can be reduced at newer NAND devices such as OneNAND™ which supports multiple block erase function. OneNAND™ can erase up to 64 blocks within constant 4ms [13].

2.3 iBAT: in-core only Block Allocation Table

MNFS proposes iBAT for both uniform write-response and robust file system. Previous file systems use I-nodes or FAT table to manage cluster based allocations. In spare area of MNFS data block, directory entry ID and block number is stored. Directory entry ID represents the file the block belongs to, and block number represents logical position in the file. FAT-like iBAT is constructed at mounting time by scanning spare area of all blocks. That is, MNFS does not maintain concentrated data structure in NAND flash storage like FAT table. This design is for uniform write-responses. In FAT file system, as file size grows, a new cluster is allocated periodically, and it takes time. In MNFS, no additional metadata modification is needed because there is no concentrated metadata for block allocation in NAND flash memory. And more this design can enhance robustness of file system because both allocation information and user data can be written by one page write.

2.4 Upward directory representation

Legacy file systems use a directory file to represent hierarchal directory structure. Directory file contains child file/directory information and it is managed by same fashion with a normal file in the file system. MNFS does not use the method. The reason is update frequency of directory file. The key assumption for MNFS design is rare updates of multimedia data. If we

use directory file method in MNFS, the assumption becomes invalid because the contents of directory file will be frequently updated even for multimedia files. Instead of it MNFS uses *upward directory representation method*. In the method, each directory entry resides in the log block has its parent directory entry ID. That is, child entry points its parent entry. For example, directory hierarchy of Figure 3 can be represented as shown in Table 1.

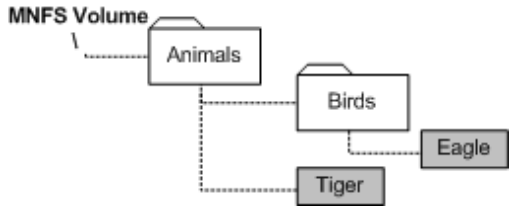

Figure 3 Example of directory representation

Table 1 Directory and parent directory entry ID

Directory Entry ID	Name	Parent Directory Entry ID
0	MNFS Volume	0
1	Animals	0
2	Birds	1
3	Tiger	1
4	Eagle	2

3. Experiments

We compared MNFS with YAFFS, and FAT file system on evaluation board which has ARM920T core processor. Main clock speed of the processor was 203 MHz and we used 1Gbits small block NAND flash memory. We used commercial FTL, XSR for FAT file system.

3.1 Write performance test

We measured write performance while we were filling clean 128Mbytes flash storage with random sized multiple files. Figure 4 shows the result. Average writing speed of YAFFS is 686 Kbytes/second and average writing speed of FAT file system is 278 Kbytes/second. MNFS is faster than other file systems. Average writing speed of MNFS is 1,538 Kbytes/seconds. YAFFS allocates memory table dynamically while processing write-requests. Because it causes heavy computational overhead, the performance of YAFFS is worse than MNFS for sequential writes even there are no garbage collections for the test.

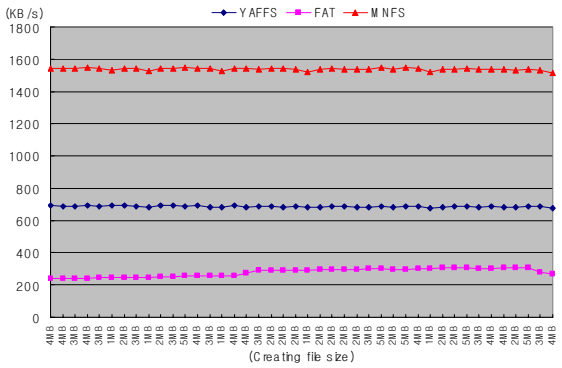

Figure 4 Sequential writing speed

3.2 File deletion test

We deleted random size files and measured processing time. Figure 5 shows file deletion time in milliseconds unit. File deletion speed of MNFS is much slower than other file systems and proportional to deleting file size because MNFS erases all blocks which are used for deleting file.

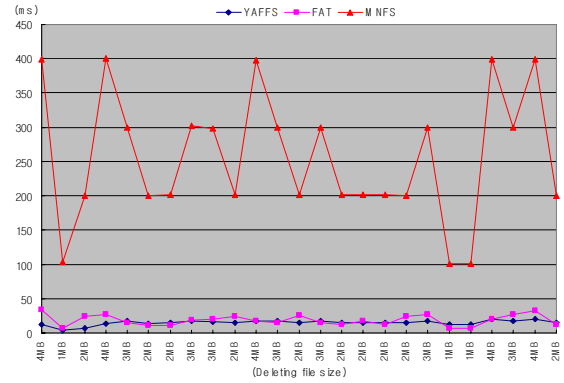

Figure 5 File deletion time

3.3 Uniform write-response test

We had created and removed random sized files to make storage fragmentation. After that, we created 64Mbytes sized file by 2,048 writes of 32Kbytes buffer. We measured each write-response time and Figure 6 shows the results. The responses of YAFFS for first 2Mbytes area were around 520ms because of garbage collection delay. After that, write-responses were uniformly 47ms. Except garbage collection, YAFFS responses are very uniform but the garbage collection is not predictable normally. The responses of FAT file system are too irregular. The longest response time was 1,832 ms.

The responses of MNFS were 21ms or 22ms uniformly. The result shows that sequential writes of MNFS are perfectly uniform.

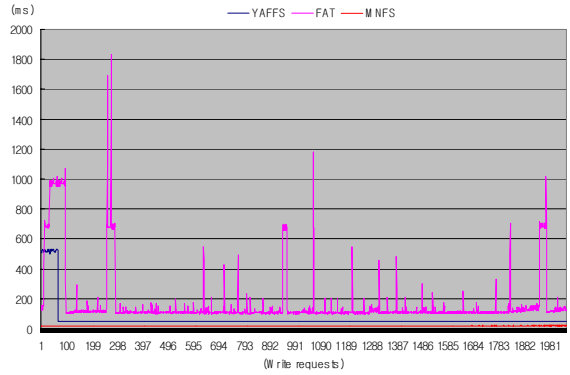

Figure 6 Response time for sequential writes

3.4 Mounting time and heap memory usage

We measured mounting time and required heap memory size when the 128Mbytes storage is full. Table 2 summarizes the results. For FAT file system we cannot measure heap memory size because we used FAT file system component of pSOS without source code. Compare to YAFFS, MNFS is 30 times faster for mounting time and the heap memory usage is only 5% of YAFFS usage.

Table 2 Mounting time and heap usages

	Mounting Time (ms)	Heap Memory Usage (Kbytes)
YAFFS	6,441	680
FAT	220	N/A
MNFS	185	34

4. Conclusion

MNFS is a new NAND flash file system for mobile multimedia devices. It is designed for predictable and uniform write latency, quick file system mount, and small memory footprint. These goals are achieved by hybrid mapping, block based file allocation, iBAT (in-core only Block Allocation Table), and upward directory representation. By experiments, we showed that MNFS can satisfy the requirements successfully.

Currently, FAT file system over FTL is widely used. However, it will not satisfy real-time requirement of mobile multimedia devices as the bandwidth of multimedia data is increasing. Because of irregular response of FTL, a large size of temporal buffer is required and it will influence the cost of devices.

MNFS can be a good solution for multimedia files. However, there may be a requirement of non-multimedia file which is small and frequently updatable. For DRM (Digital Right Management) and play list of multimedia files, non-multimedia file is required in mobile multimedia devices. For the files MNFS may inefficient for space and performance. To solve problem, we have a plan to duplicate log-structured file system over MNFS.

5. References

[1] M. Wu and W.Zwaenepoel, "eNVy: a non-volatile, main memory storage system," in Proceedings of 6th international conference on Architectural support for programming languages and operating systems, ACM Press, 1994, pp.86-94.

[2] A. Kawaguchi, S.Nishioka, and H. Motoda, "A flash-memory based file system," in Proceedings of the USNIX 1995 Technical Conference, New Orleans, Louisiana, Jan. 1995, pp.155-164.

[3] Intel Corporation, "Understanding the flash translation layer (FTL) specification," Application Note 648, 1998.

[4] P. Torelli, "The Microsoft flash file system," Dr. Dobb's Journal, Feb. 1995, pp. 62-72.

[5] M. Rosenblum, and J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol. 10, No. 1, pp. 26-51.

[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] Li-Pin Chang, Tei-Wei Kuo, "An efficient management scheme for large-scale flash-memory storage systems," in Proceedings of the 2004 ACM symposium on Applied computing, 2004, pp. 862-868.

[9] Li-Pin Chang, Tei-Wei Kuo, Shi-Wu Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," in ACM Transactions on Embedded Computing Systems (TECS), Volume 3 Issue 4, pp.837-863.

[10] SSFDC Forum, "SmartMedia™ Specification," http://www.ssfdc.or.jp

[11] Microsoft, "FAT: General Overview of On-Disk Format," Ver.1.03, 2000.

[12] Samsung Electronics, "16M x 8 Bit NAND Flash Memory," http://www.samsung.com

[13] Samsung Electronics, "512 Mb/1Gb OneNAND™ Flash Memory," http://www.samsung.com