J. Inf. Commun. Converg. Eng. 10(4): 405-410, Dec. 2012

Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 10(4): 405-410, Dec. 2012 **Regular Paper**

HFAT: Log-Based FAT File System Using Dynamic Allocation Method

Nam Ho Kim[1] and Yun Seop Yu[1,2*], *Member, KIICE*

[1]Graduate School of Bio & Information Technology and IITC, Hankyong National University, Anseong 456-749, Korea
[2]Department of Electrical, Electronic and Control Engineering and IITC, Hankyong National University, Anseong 456-749, Korea

Abstract

Several attempts have been made to add journaling capability to a traditional file allocation table (FAT) file system. However, they encountered issues such as excessive system load or instability of the journaling data itself. If journaling data is saved as a file format, it can be corrupted by a user application. However, if journaling data is saved in a fixed area such as a reserved area, the storage can be physically corrupted because of excessive system load. To solve this problem, a new method that dynamically allocates journaling data is introduced. In this method, the journaling data is not saved as a file format. Using a reserved area and reserved FAT status entry of the FAT file system specification, the journaling data can be dynamically allocated and cannot be accessed by user applications. The experimental results show that this method is more stable and scalable than other log-based FAT file systems. HFAT was tested with more than 12,000 power failures and was stable.

Index Terms: Dynamic allocation, Journaling, Log-based FAT file system, Power fail safe, Recovery

I. INTRODUCTION

With the development of embedded systems technology, many mobile devices have used mass storage devices and file systems for efficient use of data. Because the file system used on mobile devices should be robust against sudden power failures and system crashes, the use of a journaling file system is needed [1, 2]. Journaling has been mainly used in network server technology [1, 2]. This technology needs data reliability in critical circumstances such as power outages and abnormal termination. Mobile devices also need such reliability to increase their consistency. In addition, mobile devices need to prevent data corruption due to abnormal termination and slow boot time.

Removable storage devices such as USB memory, SD cards, and T-flash, widely used in mobile devices, must use the file allocation table (FAT) file system for windows PC compatibility. However, the FAT file system does not contain journaling, and thus many problems with the consistency of data occur. Therefore, to enhance the consistency of data in removable storage devices with the FAT file system for Windows operating system (OS)-based PC compatibility, several methods have been reported [3-5]. The performance of Microsoft® TFAT [3] is relatively poor because the entire FAT area is backed up whenever the commit occurs. Samsung® KFAT [4] can be damaged easily because the journaling data can be accessible to the user. TI-LFAT's method is limited in expansion of the journal data's size [5].

In this paper, a new method for managing journaling data is introduced. Using the reserved area and reserved FAT entry, the location and quantity of journaling data can be

Received 18 July 2012, **Revised** 30 July 2012, **Accepted** 13 August 2012
***Corresponding Author** E-mail: ysyu@hknu.ac.kr

Open Access **http://dx.doi.org/10.6109/jicce.2012.10.4.405** print ISSN: 2234-5973 online ISSN: 2234-8883

dynamically allocated. The rest of this paper is organized as follows. Section II presents the background for this research and works related to this paper, and Section III describes the structure of the log data and design of Hankyong Journaling FAT (HFAT) used in this paper. In Section IV, the test environment is illustrated and a performance comparison among the implemented HFAT and other approaches is explained. Finally, we provide a summary and conclude in Section V.

II. BACKGROUND AND RELATED WORKS

Storage devices used in embedded systems have often used only a small amount of data. However, with the development of hardware and software technologies such as capacity in recent years, mobile devices can contain storage devices from hundreds of megabytes to several gigabytes.

In order to effectively use the mass storage devices, it is necessary to use a file system. The specific file system used mainly depends on the kind of OS or storage medium. For example, if you use NAND memory on Linux or Android, YAFFS [6] or JFFS2 [7] will be used. Similarly, the exFAT [8] or transaction-safe FAT (TFAT) [3] file system is used in Windows CE (WinCE). However, these kinds of file systems cannot be used in removable storage memory, but in fixed NAND flash memory. In order to run directly on a Windows OS-based PC, a file system should be compatible with the FAT file system. However, the FAT file system does not include a journaling feature. Therefore, several proposals for a FAT file system design with a journaling feature have been reported [3-5].

A. Reserved Area and Reserved Cluster Value of Traditional FAT File System

To explain the structure of journaling, the traditional FAT file system is described as follows. The approximate structure of the traditional FAT file system is shown in Fig. 1. FAT16 usually has a reserved area consisting of just one sector to save the boot record. In contrast, FAT32 assigns 32 sectors as a reserved area.

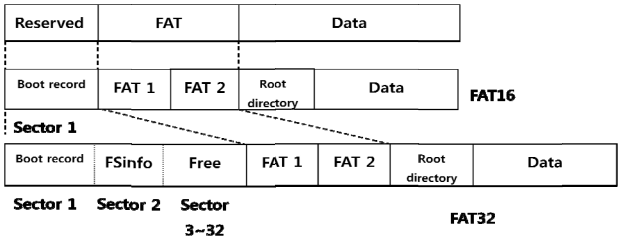

Fig. 1. The structure of FAT16/32 status. FAT: file allocation table.

The first sector is the boot record such as FAT16. The second sector is the file system information (FSInfo) which knows how much free space can be used and where the next free cluster is. Space also exists behind the 30 free sectors. Next to the reserved area is the FAT area, which represents the connection of a cluster data structure as a single linked list. The following Table 1 shows the meaning of the FAT entry [9]. A normal file or directory begins in the allocated cluster and ends in the end of cluster chain (EOC).

B. Related Work

There have been several attempts to add a journaling capability to a FAT file system. TFAT [3], which is employed in WinCE 6.0, has been developed by Microsoft. Originally, the FAT area consisted of FAT1 and FAT2, and FAT1 is the same as FAT2. Typically, FAT2 is set as active working FAT1. All FAT entries are first updated in FAT2 and then the entire FAT2 is copied to FAT1 during the commit. The file/directory updates are performed by allocating a new data cluster and then writing the updated data instead of updating the existing data cluster. This causes additional writes to FAT entries. The root directory updates of FAT12 and FAT16 are not safe transactions. This is the reason why the root directory in these file systems exists in a fixed location [3]. If one operation is completed, the entire FAT2 area is copied to FAT1. For this reason, performance degradation is severe.

Kwon et al. [4] proposed a new algorithm called KFAT. Unlike TFAT, this method logs all changes such as those of the FAT entries and directory/file entries. During the commit operation, the logged entries are processed, and finally FAT1, FAT2, and directory entries are updated. Typically, the log file is the first file in the root directory and its size is fixed. It contains 13 different log types. RFS [10] and TFS4 [5] are based on KFAT. TFS4 is specific to Samsung Electronics oneNAND™ flash memory. The disadvantage of this method is that if journaling is saved as a file format it can be corrupted by a user application. The amount of log data is smaller than the TFAT, but the number of access time exceeds the TFAT.

Table 1. FAT entry status values

FAT16	FAT32	Explanation
0x0000	0x?0000000	Free cluster
0x0001	0x?0000001	Reserved cluster
0x0002–0xEEEF	0x?0000002–0x?FFFFFEF	Allocated cluster
0xFFF0–0xFFF6	0x?FFFFFF0–0x?FFFFFF6	Reserved cluster
0xFFF7	0x?FFFFFF7	Bad cluster
0xFFF8–0xFFFF	0x?FFFFFF8–0x?FFFFFFF	End of cluster

FAT: file allocation table.

Table 2. Problems of traditional FAT file system

Operation	Access area and sequence*	Explanation of operation	Problems
open (create)	DIR entry	Generate file name, creation time, write hour, cluster start number, size 0 of entry	
write	FAT→Data	Save the FAT first and store the actual data	If power off occurs during saving the FAT, isolation of the FAT will occur.
close	DIR entry	Save write time and file size.	If file size is not up-dated, isolation of the FAT will occur
remove	DIR entry →FAT	Write E5 in the directory entry and erase FAT	If power off occurs during erasing the FAT, isolation of the FAT will occur
mkdir	Present DIR entry →FAT →new cluster clean →lower DIR entry	Create a new directory entry for the FAT, and clean a cluster to make space for the new cluster, and create a sub-entry	If power off occurs before creating a sub-directory entry, a new file or directory will be generated
rmdir	DIR entry(E5) →FAT	Write E5 in the directory entry and erase FAT.	If power off occurs before saving the FAT, isolation of the FAT will occur
rename	DIR entry →DIR entry(E5)	Create a new entry and store E5 in the existing entry	If a new entry is only created and the existing entry is not erased, the existence of two files with same start cluster will occur
set attribute	DIR entry	Save directory entry to change property	
truncate	DIR entry →FAT	Modify file size in the entry, and erase FAT	If size information is saved and FAT is not erased, the existence of a file with no end of file will occur

*DIR entry represents the directory entry area, and FAT represents FAT area, and Data represents data area.

Munegowda [5] at Texas Instruments proposed a new method to take advantage of the KFAT and TFAT. FAT entries logging is similar to TFAT. However, the entire FAT2 is not completely copied to FAT1. Only updated FAT2 entries are copied to FAT1. Directory entries logging is similar to KFAT. FAT entry updates are not logged in the log file, but only 32 byte directory entry updates are logged. This method has the disadvantage that the log file is exposed to user applications.

C. Problems of the Traditional FAT File System

In the traditional FAT file system with no journaling, the file operations such as file create (open), write, close, remove, rename, set attribute, make dir, remove dir, and truncate can cause some problems, as shown in Table 2. Each file operation accesses a different area and sequence. E5 is an erase operation on a file or directory name. If 0xE5 is written at the beginning of its entry, this entry will be ignored.

III. MANAGEMENT OF LOG DATA FOR HFAT.

Section III-A describes an algorithm which manages the reserved area and FAT area to allocate dynamically the location and size of log data, and it can be understood as the core of this algorithm. Section III-B describes how to implement HFAT.

A. The Cluster Number of the Log Data

Journaling of HFAT is not saved as a file format as it is in KFAT. HFAT use the reserved area and the reserved cluster number of the FAT entry, as shown in Table 1. The start cluster number of the log data is saved in the next FSInfo sector of the reserved area. As shown in Table 1, the FAT entry status value of the reserved cluster for FAT32 is 0x?FFFFFF0–0x?FFFFFF6.

Fig. 2 shows the cluster number of the log data using the FAT entry status. The upper 4 bits of the FAT entry status value are reserved, and it does not have any effect. And the lower 4 bits of the value 0x0–0x6 is can be used. 0x0–0x5 of the lower 4 bits is used as the upper 4 bits of cluster number and 0x6 is defined as end of cluster (EOC). The upper 4 bits of FAT entry status value is used as the lower 4 bits of the cluster number.

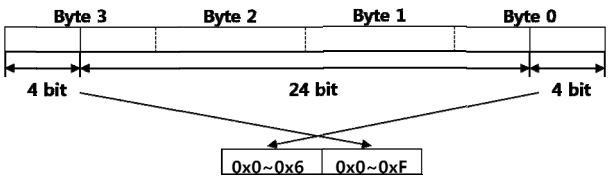

Fig. 2. The cluster number of log data using file allocation table (FAT) entry status.

In other words, 0x00–0x5F can be assigned to the cluster number. A total of 95 clusters can be used as log data.

The length of the cluster chain can vary within the range of 0x00–0x5F, and the number of cluster chains can increase to more than one. In other words, the cluster number of the log data can allocated dynamically.

Fig. 3 shows an example of using the value of the FAT table for the log cluster chain. In this example, there are two cluster chains. The start cluster number of the first log data is 0x1000 and the second is 0x5000. The corresponding FAT entry of the first cluster chain status value is 0x1FFFFFF0 and that of the second is 0x0FFFFFF2. According to Fig. 2, 0x1FFFFFF0, 0x3FFFFFF0, and 0x0FFFFFF6 represent 0x01, 0x03, and 0x06 (EOC), respectively. In conclusion, the cluster chain of the first log data is 0x1000 → 0x1001 → 0x1003 → 0x60 (EOC), because the starting cluster of the log data is 0x1000. The second log data is 0x5000 → 0x5020 → 0x60 (EOC). This cluster chain cannot be used for general files or directories because these status values are reserved. Using this method, the total storage of the log data can be extended up to 95 clusters, unlike the size and location of the log data in KFAT or TI-LFAT, which remain fixed.

B. Implementation of HFAT

HFAT can be divided into a FAT core module, journaling module, OS abstraction layer (OAL), and hardware abstraction layer (HAL). The descriptions of each module are as follows:

1) FAT core: It performs the original function of the FAT file system. At boot time, it mounts a file system. Then, it manages the FAT table, directory entry, and file data. It can be divided into four modules; mount manager (MM), FAT table manager (FTM), directory entry manager (DEM), and cache manager (CM).
2) Journaling core: It performs the function of producing journaling data and recovering the file system if needed. Depending on the capabilities, it can be divided into three modules; journaling data region manager (JDRM), journaling recovery manager (JRM), and journaling production manager (JPM).
3) OAL: OAL represents the OS abstraction layer. HFAT follows the portable operating system interface (POSIX) standard application programming interface (API).
4) HAL: HAL represents the hardware abstraction layer. It depends on the OS layer.

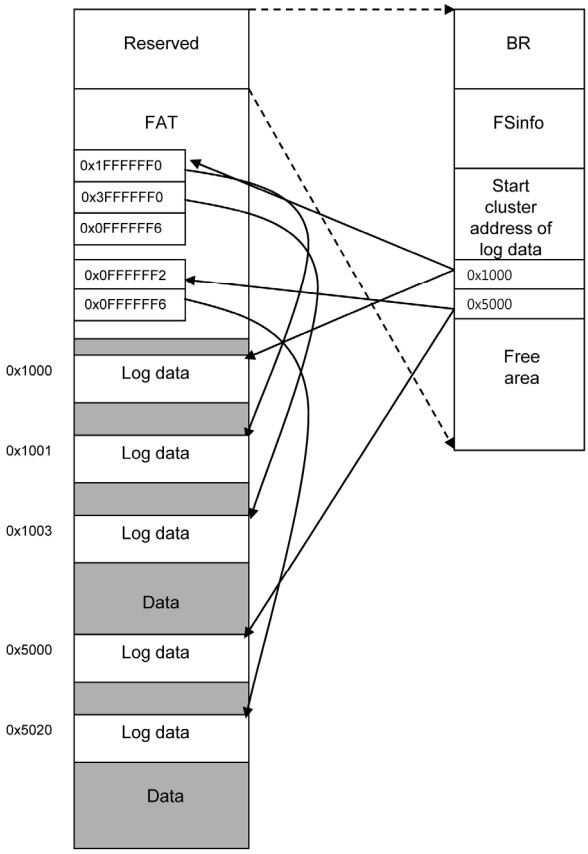

Fig. 3. Example of cluster number of log data. FAT: file allocation table, FSinfo: file system information.

Fig. 4 illustrates the relationships among the modules and the sequence of HFAT operations. HFAT is mainly divided into the FAT core and journaling core. At boot time, the MM obtains data information such as the log data address, FAT table address, data region address, and sector size. The MM transfers this information to the FTM, DEM, and JDRM. If the JRM finds an unfinished log during mount time, it performs data recovery. The FTM and DEM pass the FAT table and directory entry to the JPM. The JPM generates the journaling data based on this information.

IV. EXPERIMENTS AND RESULTS

We implemented the HFAT file system and tested it on the mango6410 board [11] on WinCE 6.0. This development board is Samsung S3C6410 ARM11-based mobile device, which has NAND flash and SD/MMC interfaces. The clock speed was 667 MHz. Fig. 5. Shows mango6410 board is used in our experiments. The transactional property of HFAT was tested in the following methods.

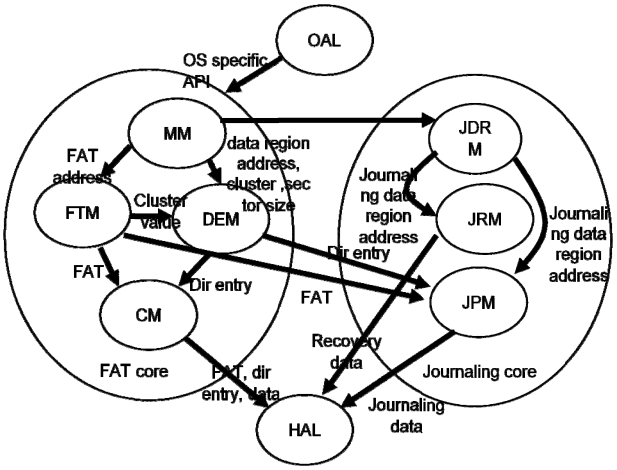

Fig. 4. Configuration and association of each module for Hankyong journaling FAT (HFAT). FAT: file allocation table, API: application programming interface, OAL: operating system abstraction layer, MM: mount manager, FTM: FAT table manager, DEM: directory entry manager, CM: cache manager, JDRM: journaling data region manager, JRM: journaling recovery manager, JPM: journaling production manager, HAL: hardware abstraction layer.

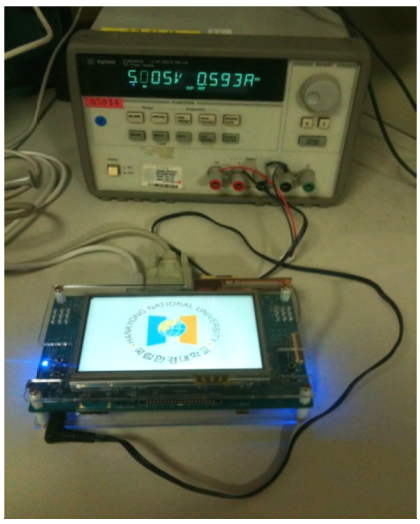

Fig. 5. Picture for test environment.

The automatic power on/off device is used with the DC power supply (Agilent™ E3633A) controlled by National Instrument™ GPIB device. The interval of power off is generated randomly and usually occurs once every 5–10 minutes. The experiment was performed for 3 months. As a result, there were more than a total of 12,000 on/off power tests. If a problem occurred when mounting the file system, the system would stop. Of course, the degradation of performance using the HFAT occurs naturally. However, on WinCE, it depends mainly on the disk cache policy rather than journal data saving. When we used WinCE disk cache, there was about 15% degradation of performance. However, the self-made cache with the 4 kB disk cache per volume allows direct write operation. In this case, there was no significant difference in performance degradation.

V. CONCLUSIONS

To solve the problems of the traditional FAT file system, a new method of journaling with the FAT file system was proposed. Because log data is not saved as a file format, user applications cannot access the log data directly. This will increase the reliability of log data. Because the size of log data can expand to 95 clusters, this is more scalable than other log-based FAT file systems. This method can be used in managing security data as well as log data. In the future, it will be implemented on Android and Linux.

REFERENCES

[1] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems," in *Proceedings of the Annual Technical Conference on USENIX*, Anaheim: CA, p. 8, 2005.

[2] C. Hyun, J. Choi, D. Lee, and S. H. Noh, "Temporary metadata journaling scheme to improve performance and stability of a FAT compatible file system," *Journal of KISS: Computer Systems and Theory*, vol. 36, no. 3, pp. 191-198, 2009.

[3] Microsoft Corp., TFAT overview [Internet], Available: http://msdn.microsoft.com/en-us/library/aa915463.aspx.

[4] 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 US-Korea Conference*, ICTA-142, 2005.

[5] K. Munegowda, Power fail safe FAT file system [Internet], Available: http://elinux.org/images/5/54/Elc2011_munegowda.pdf.

[6] Aleph One Ltd., Yaffs original specification version 0.3 [Internet], Available: http://www.yaffs.net/yaffs-original-specification.

[7] D. Woodhouse, JFFS: the journaling flash file system [Internet], Available: http://www.sourceware.org/jffs2/jffs2.pdf.

[8] Microsoft Corp., Extended FAT file system [Internet], http://msdn.microsoft.com/en-us/library/aa914353.aspx.

[9] Microsoft Corp., Microsoft EFI FAT32 file system specification [Internet], Available: http://msdn.microsoft.com/en-us/windows/hardware/gg463080.aspx.

[10] Wikipedia, RFS [Internet], Available: http://ko.wikipedia.org/wiki/RFS.

[11] Mango6410 board [Internet], Available: http://cafe.naver.com/embeddedcrazyboys.

Nam Ho Kim
received B.S. and M.S. degrees from the Department of Electronics Engineering of Korea University, Seoul, Korea, in 1996 and 1998, respectively. He is currently pursuing a Ph.D. in the Graduate School of Bio & Information Technology at Hankyong National University, Anseong, Korea. His main research interests are embedded systems, file systems, and device drivers of various OSs. He is also interested in face and gesture recognition

Yun Seop Yu
received B.S., M.S., and Ph.D. degrees from the Department of Electronics Engineering of Korea University, Seoul, Korea, in 1995, 1997, and 2001, respectively. From 2001 to 2002, he worked as a Guest Researcher in the Electronics and Electrical Engineering Laboratory at NIST, Gaithersburg, MD. He is now an Associate Professor with the Department of Electrical, Electronic and Control Engineering, Hankyong National University, Anseong, Korea. His main research interests are in the fields of the modeling of various nano devices for efficient circuit simulation, and future memory, logic, and sensor designs using those devices. He has authored and coauthored 50 refereed international journal papers. He is also interested in the fabrication and characterization of various nano devices, as well as those applications such as future memory, logic, and sensors.