Design and Implementation of Log Structured FAT and ExFAT File Systems

Keshava Munegowda [#1], Dr. G T Raju [#2], Veera Maninkandanraju [#3]

[#1]Principal Software Engineer,
Advanced Storage Division,
EMC Corporation
Bangalore, INDIA
keshava.gowda@gmail.com
keshava.munegowda@emc.com

[#2]Professor and Head,
Department of computer Science and Engineering, RNSIT,
Bangalore, INDIA
gtraju1990@yahoo.com

[#3]Senior Member Technical Staff,
Texas Instruments
Bangalore, INDIA
veera@ti.com

***Abstract*— The File Allocation Table (FAT) file system is supported in multiple Operating Systems (OS). Hence, FAT file system is universal exchange format for files/directories used in Solid State Drives (SSD) and Hard disk Drives (HDD). The Microsoft Corporation introduced the new file system called Extended FAT file system (ExFAT) to support larger size storage devices. The ExFAT file system is optimized to use with SSDs. But, Both FAT and ExFAT are not power fail safe. This means that the uncontrolled power loss or abrupt storage device removable from the computer system, during file system update, causes corruption of file system meta data and hence it leads to loss of data in storage device. This paper implements the Logging and Committing features to FAT and ExFAT file systems and ensures that the file system meta data is consistent across the abrupt power loss or device removal from the computer system.**

Keyword- Commit, Directory, ExFAT, File System, FAT, FATTY, Flash Memory, HDD, HFAT, Journaling, KFAT, Logging, Log Structure, MMC, Power Fail-Safe, RFS, SD, SSD, TFAT, TexFAT, TFS4, TI-LExFAT, TI-LFAT.

I. INTRODUCTION

The FAT file system was initially developed to use on floppy disks and hard-drives. The earlier version of File Allocation Table (FAT) [1] file system was FAT12 by Microsoft Corporation, later it was extended as FAT16 and further as FAT32 to support higher storage size. Since most of the operating systems such as Linux, windows, UNIX and Mac OS implements the FAT file system, this file system has become a default and universal exchange file storage format for embedded devices. Today, The File Allocation Table (FAT) file widely used in embedded storage devices such as MMC [2]/ SD [3] Micro SD cards, NOR, NAND flash memories [4], USB (Universal Serial Bus) pen drives and many more. The MMC/SD storage cards, USB pen drives are categorised as "Removable storage devices". Even though FAT file system does not define flash management techniques such as wear-levelling and Bad Block management, the embedded storage devices implements this file system along with the dedicated Flash Translation Layer (FTL) [5]. In FAT file system, the file or directory is the linked list of the clusters. A cluster is a group of sequential blocks or sectors of storage device. The File Allocation Table contains the linked list of clusters of files/directories. As per Specification, the maximum storage size supported by FAT32 file system is 32 GB (Giga Bytes). But, theoretically, FAT file system can support up to 16 TB (Tera Bytes) of storage device with 128 sectors per cluster and 512 bytes of sector size. The FAT file system was the only file system used during Disk Operating System (DOS), windows 95/98 operating systems. Later, Microsoft introduced the NTFS (New Technology File System) [6] [7] in the Windows NT operating system to support higher storage size. Due to write buffering/caching mechanism in NTFS causes the file system corruption when the flash devices are used as removable storage devices with the desktop PCs or with any other embedded device. The NTFS has the security features which are desktop user specific and optional for the embedded storage devices. The write caching mechanism of NTFS file system causes the data corruption when the removable storage devices such as MMC/SD, USB pen-drives are unplugged abruptly. Due this Microsoft introduced the new file system Extended FAT (ExFAT) [8] [9] file system for the flash devices. The ExFAT file system overcomes the storage size limitation of FAT32 file system by addressing the storage size up to 128 PB (Peta Bytes). The ExFAT file system simplifies the user attributes of the files and directories to ease the usage with flash storage devices. The ExFAT does not use the write cache

mechanism for the removable storage devices. The FAT and ExFAT file systems are not power fail safe. This means, if the uncontrolled power loss or device removal during file system write/update operation, then it causes incomplete updates to file system meta data. Such update makes file system data is unreliable and enforces the file system user to format the storage devices and hence losses all previously stored valid user data in the storage device. The FAT file system operation except the read operation involves following operations

1. The Meta data update involves
    a) Updating the 32 byte directory entry [1] of file/directory and
    b) Updating the File Allocation Table.
2. The User data update involves the updating the data clusters of the file.

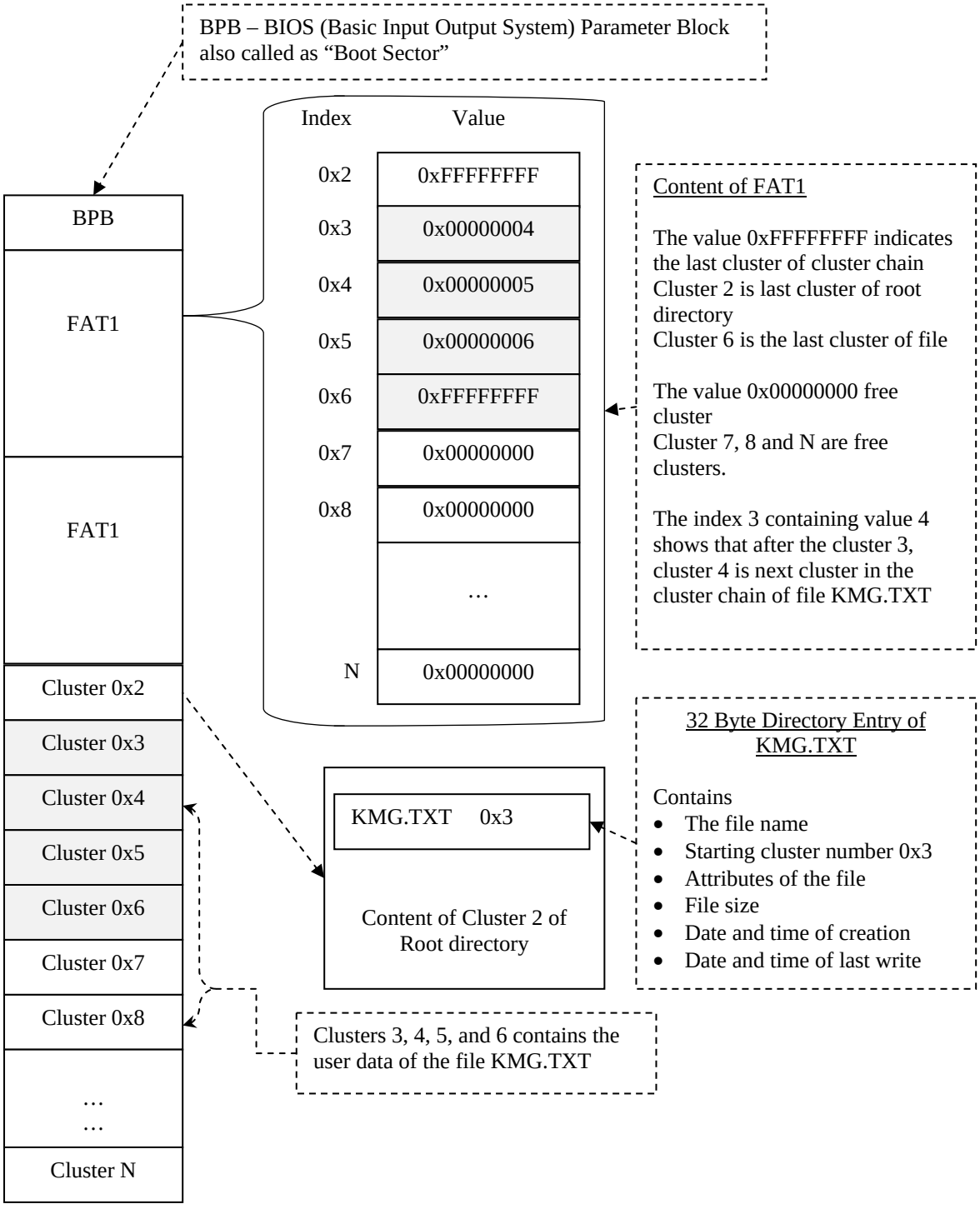

Fig.1. Organisation of file KMG.TXT in FAT32 files system

Figure 1 shows the example file in FAT32 file system. The file named KMG.TXT exists in the root directory with the clusters numbered 3, 4, 5, and 6 allocated. The File Allocation Table show the linked list of the cluster allocated to the file. The cluster 2 of the root directory contains the 32 byte directory entry of the file. Whenever the file update such as writing a new data, updating the existing data or truncating the file is performed, the 32 byte directory is updated with last accessed time and new file size. While allocating or removing the clusters from the file, the status of the cluster is updated in FAT. If there is an uncontrolled power loss or abrupt storage device removal from computer/embedded system during cluster status (FAT entry) update in File Allocation Table or during update of 32 byte directory entry , then it causes the file system corruption and hence it cause unreliability of the existing data of the file system. In the example shown in Figure 1, if there is uncontrolled power loss while updating the FAT or 32 byte directory entry of KMG.TXT in the cluster 2, then it corrupts the file system. During user data write to the clusters of the file, if there is uncontrolled power loss or sudden storage device removal then it cause the corruption of the file data and user loses the content of the file. The uncontrolled power loss or sudden device removal from the computer/embedded system during a meta data update cause corruption of entire file system which is much more serious problem than losing content of a file. In the example shown in Figure 1, if there is uncontrolled power loss while writing data to the clusters 3, 4, 5 and 6 then it cause the unreliability of the data of the file KMG.TXT.

This paper implements the file system TI-LFAT (Texas Instruments Log Structured FAT file system) [10]. The TI-LFAT is the extension of FAT file system with power fail safe feature. The Logging/journaling, committing and crash recovery algorithms of TI-LFAT were first proposed and published in Embedded Linux Conference (ELC), San Francisco, California, United States of America, 2011. Along with TI-LFAT file system, this paper defines and implements another new file system TI-LExFAT (Texas Instruments Extended Log structured FAT file system). The TI-LExFAT is the extension of ExFAT with power fail safe feature. This paper defines common logging, committing and crash recovery algorithms for both TI-LFAT and TI-LExFAT. This paper compares the design aspects of the TI-LFAT and TI-LExFAT file systems with existing power fail safe FAT and ExFAT file systems. The implementation of the TI-LFAT and TI-LExFAT maintains the backward compatibility with FAT and ExFAT file system specifications and implementations. This means the file/directories created any of TI-LFAT and TI-LExFAT file systems are accessible for read and write operations in FAT and ExFAT file system implementations respectively.

II. RELATED WORK

The concept of Logging/Journaling the Meta data and user data of the file system is not a new idea. The LFS [11] [12] was introduced in 1990s uses the Log structures to log the UNIX file system update operations and file system is modified during commit operation. Later, the journaling mechanism was introduced in ext2 [13] file system with some more improvements and referred as ext3[14] file system in Linux operating system version 2.4 and later versions. The later versions of Linux 2.6 kernels included the Ext4 [15] file system which addresses the larger storage devices. Since journaling feature requires additional block/sector writes to storage device and it reduces the file system performance, the Ext4 file system has the flexibility to enable and disable the journaling feature. The new Btrfs [16] file system is gaining popularity among Linux file systems. The Btrfs file system provides journaling, easy repair and administration features. The Ext series file systems and Btrfs file systems are used in hard-drives and can be used in MMC/SD cards and USB pen drives, but these file systems are not applicable for raw NAND/NOR flash memories because flash memory specific wear-levelling and garbage collection algorithms are not defined. The LFS was found suitable for flash memories because flash memories used with embedded devices which are depended on batteries for power and flash memory based storage devices such as MMC/SD, USB pen drive etc. are used as “removable storage device” in computer system or embedded systems. Hence, a variant of LFS called JFFS (Journaling flash file system) was designed for raw NOR flash memories with the performance optimizations. Later The JFFS v2.0 [17] was released with NAND flash memory support. Since JFFS file system has the longer mount time (initialization time), to avoid this LogFS (Log structured and scalable file system) [18] was introduced. LogFS stores the inode tree of file system on the drive; JFFS2 does not, which requires it to scan the entire drive at mount and cache the entire tree in RAM (Random Access Memory)/ Main memory. For larger drives, this scan can take tens of seconds and the tree can take a significant amount of main memory. LogFS avoids these penalties, but it does do more work while the system is running and uses some of the storage space for holding the inode tree. In 2007, Nokia introduced the new file system called UBIFS (Unsorted Block Image File System) [19]. The UBIFS is also a successor of JFFS2 and a competitor to LogFS. Like LogFS, the UBIFS file system also improves the mounting and write speeds. But the UBIFS file system is applicable only for raw NAND flash memories and not applicable for MMC/SD cards. The UBIFS allows the on the fly compression algorithm while writing data to flash. The YAFFS (Yet Another Flash File System) [20] is another log structured flash file system with higher data integrity and performance for NAND flash memories. Recently the Samsung has introduced a new file system called F2FS (Flash Friendly File System) [21]. The F2FS file system is also a log structured file system

taking in to account the characteristics of both raw NAND flash devices and MMC/SD storage and USB pen drives.

Due to Multiple OS support of FAT File system, there are several variants of FAT file system with power fail safe feature are developed. The KFAT[22], Robust File System (RFS) [23], Transaction File system (TFS) version 4[24], Device and Method for Data Recovery in file system[25], File system for avoiding loss of data[26], Journaling FAT [27] , HFAT[28] file systems are the variants of the FAT file system. These file system achieves the power fail safe file system update/write operations by logging the file system meta data to be updated and optionally logs the file system user data too. Each of these file systems differs in the location of journal or log records are placed in the storage device. The KFAT, RFS and TFS4 file systems are developed by Samsung. These file systems maintains the file system updates in log file. The TFS4 uses KFAT file system for logging and crash recovery and it provides File system abstraction layer for the applications. The RFS file system is optimal for OneNAND flash memories. The log file of these file systems are visible to user of file system. In case abrupt power failures, the recovering the file system data during next system boot using Non Volatile Memory is defined in Device and Method for Data Recovery in file system [25]. In this method, the recovery information is not in form of file and it not part of the file system. Hence, this method is not suitable for removable storage devices. The "reserved sectors" of FAT file system are used to store the file system log is defined in File system for avoiding loss of data [26]. In this method, the file system updates are first logged / written in to the reserved sectors of FAT file system and then it is updated with file system. In this method, the file system logs are not stored in the form of file and logging information is hidden from the file system user and it avoids accidental updates of the log. The number of reserved sectors to store the file system log needs to be determined during file system format operation. To create the reserved sectors of FAT file system, User of the file system need to format the storage devices and it erases all data of the file system. Journaling FAT file system stores the file system log in separate partition of the storage device. In this method also, the file system logs are not stored in the form of file and logging information is hidden from the file system user and it avoids accidental updates of the log. But, this approach also requires format operation to create the multiple partitions with at least one partition dedicatedly used for file system logging and recovery. This method not applicable if there is no non-file system partition in the storage device. The HFAT file system is the extension of the TI-LFAT file system. The TI-LFAT uses the reserved clusters [29] of the FAT file system to store the logging information of the file system updates. The advantages of using reserved clusters store the file system log is the logging information is not in the form of file and hence it is hidden from user. The reserved clusters can be created in already formatted file system containing file/directories. Whereas, create the "reserved sectors" in FAT file system requires file system format operation which not required to create the "reserved clusters" in FAT file system. If the journaling/logging is not required for FAT File system, the reserved clusters can be converted as free clusters which can used to store the file/directory data. The TI-LFAT file system reserves the required and limited number clusters to log the file system meta data updates. If the is no free space in the reserved clusters to store the log, then free space is created by committing the logs to file system. In HFAT file system, if the is no space in the reserved clusters to store the log, then additional space is created by making any available free cluster as reserved cluster to use for the logging of file system operation. Hence, the in HFAT file system the reserved cluster area dynamically expands as the number of uncommitted logs increases. The TFAT [30] is the extension of FAT file system and TexFAT [31] file system the extension of ExFAT file system. The TFAT and TexFAT file systems are available in Microsoft's Windows CE 6.0 and later versions windows embedded operating systems. The TFAT and TexFAT file systems provides the power fail safe feature. The TFAT File system does not maintains any log information of the file system operation; instead FAT2 is updated first with the requested file system operation. Upon Successful completion of transaction, the FAT2 is copies to FAT1. If the power-failure occurs while updating FAT2, then the contents of FAT2 are completely ignored and FAT1 is copied to FAT2, thus consistency of the File system is maintained. But, the TFAT is very slow in performance and it also has limitation that any write operation performed on the root directory of FAT12/16 is not transaction safe. The TexFAT file system uses transactions safe cluster allocation strategies of the TFAT and hence it has lower file system performance. The KFAT, RFS, TFS4, Device and Method for Data Recovery in file system [25], File system for avoiding loss of data[26], Journaling FAT does not define the log structures and algorithms for file system logging, committing and crash recovery. The KFAT file system states that it logs both 32 byte directory entry updates and FAT updates. This reduces the file system performance. The KFAT file system claims that average file write performance reduction is 12%. The FATTY [32] file system proposes prototype of Delayed Sequential Write (DSW) of file system updates instead of logging operation. But, FATTY file system does not claim the full support of reliability of FAT file system. All these power fail safe FAT file systems are backward compatible with conventional FAT file system. This paper details the complete design of TI-LFAT and TI-ExFAT file systems. This paper also records the performance benchmarking of implementations of TI-LFAT, TI-LExFAT in comparison with FAT and ExFAT file systems respectively.

III. TI-LFAT: TEXAS INSTRUMENTS LOG STRUCTURED FAT FILE SYSTEM

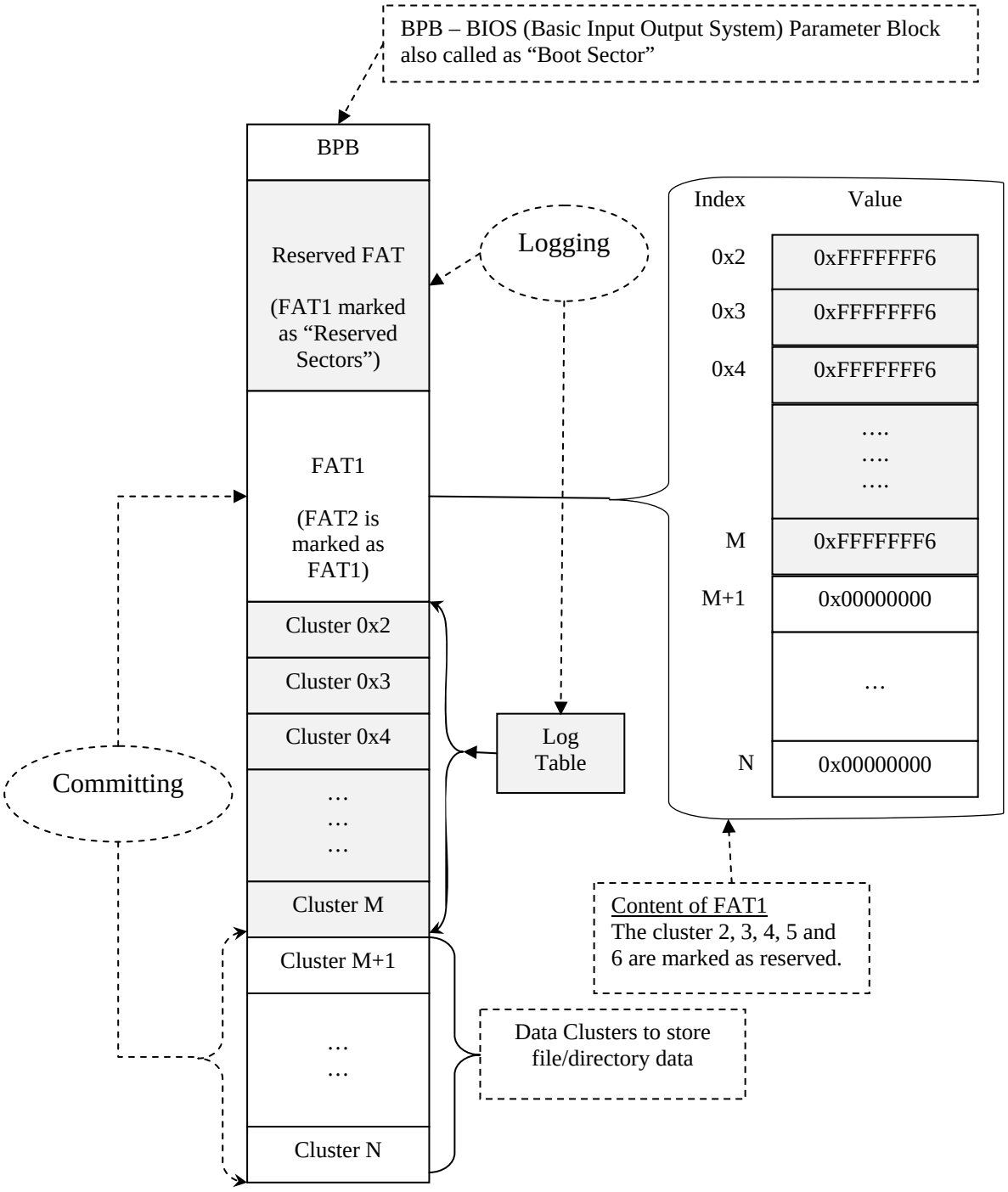

Fig.2. Organisation TI-LFAT File System

The TI-LFAT file system defines the "Logging" and "Commit" operations follows

- *Logging*

The TI-LFAT logs the 32 bytes directory entry updates and same are the stored in the form of "Log Records" in the "Log Table". The Log Table can be placed in a normal file in root directory of the FAT file system which is same as KFAT, RFS and TFS4 file systems. The Log Table can be placed in the "reserved sectors" which is same as technique discussed in File system for avoiding loss of data [26]. The TI-LFAT file system stores the "Log Table" in reserved clusters [29]. To create the Log Table, the File Allocation Table is search for contiguous required number of free clusters and these free clusters are marked as reserved clusters.

The TI-LFAT does not log the "FAT Updates" instead FAT1 is converted as "Reserved sectors" and FAT updates are first written into these reserved sectors and the updated sectors of the reserved sectors are copied to File Allocation Table during "commit operation". The area of reserved sectors used as FAT is called "Reserved FAT". This technique is similar to TFAT file system. During file system format, the sectors occupied by FAT1are marked as reserved sectors and FAT2 is used as FAT1. If the file system is already containing valid data and format operation not desirable then during file system initialisation or mount operation the FAT1 sectors are add to the reserved sector count as described in the method Dynamic reduction of File Allocation Tables [33]. If there is an uncontrolled power loss or abrupt device removal during logging, then such file system operations are discarded. If log records are written in to Log table and while committing if there sudden power loss or abrupt device removal from computer/embedded system, then during reboot of the system or during mounting of the file system, the log records are committed to the file system.

- *Committing*

During file create, close, rename, and delete operations, the "Log records" containing the 32 byte directory entry updates are processed and the updates are made the file system. After committing a log record to the file system, the log record is marked as "committed". The updated sectors of the "Reserved Fat", containing the FAT updates are written in to the File Allocation Table. During every boot and file system initialization and mount operation, the Log records are parsed and if there are any uncommitted log records then operation specified by such log records are performed to the file system and same log records are marked as "committed".

Figure 2 shows the organisation of TI-LFAT file system with N data clusters. As an example, Note that clusters 2 to M (M < N) are marked as reserved clusters in which "Log Table" is placed; The TI-LFAT uses only one File Allocation Table. The Update of FAT is first written to "Reserved FAT" and then it is committed to FAT1. Figure 3 shows the logical organisation of "Log Table". Table I shows the structure of the "Log Record". Each Log Record is of 32 bytes size and it similar to 32 byte directory entry. A single file system operation requires a single or multiple log records. The file system operation such as file/directory creation with SFN (Short File Name) [1], file write, file/directory delete and file truncation requires one log record. The File system operations such as file/directory creation with LFN (Long File Name) [1], file rename requires multiple log records. Table II shows the log record with LFN. Since, FAT file system supports maximum of 256 characters of LFN, maximum of 16 LFN Log records per file system operation is permitted in Log Table. Figure 4 depicts the combined flow chart of the Logging and committing operations. The DIR_LogType of Log Record is reset to 0 to indicate that the log record is committed.

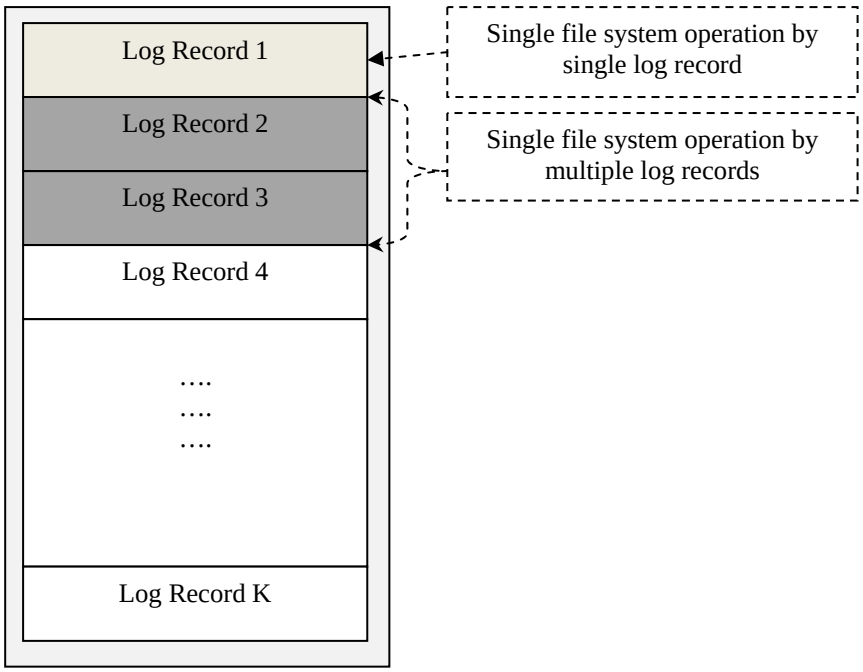

Fig. 3. Content of Log Table

TABLE I
Structure of 32 Bytes Log Record of TI-LFAT File System

Name	Offset	Size in Bytes	Description
DIR_LogType	0	1	Type of the File system Operation Value: File system Operation 1: File/Directory Create 2: File Write 3: File Truncate 4: File/Directory Delete 5: File/Directory Rename 6: Contiguous cluster chain allocated 7: File system operation of previous Log Record; "Sub Log Record" and it is part of multiple log record to perform single file system operation. 0: Log Record is committed
DIR_LFNCount	1	1	Long File Name (LFN) Entries Count
DIR_Name	2	11	Short File Name
DIR_Attr	13	1	File/Directory Attributes
DIR_DIndex	14	2	Index in the DIR_Dcluster; This index points to 32 bytes directory Entry
DIR_DCluster	16	4	Parent directory Cluster; The cluster in which this 32 bytes directory entry exists or to be created
DIR_FCluster / DIR_LCluster	20	4	This value can contain the either first cluster or last cluster of the file/directory based on the DIR_LogType operation. DIR_LogType: Description File Write: Last Cluster of the 32 byte Directory Entry; DIR_LCluster is the last cluster of the cluster chain of the file before the write operation. File/Directory Create, File Truncate, File/Directory Delete, File/Directory Rename: First Cluster of the 32 byte Directory Entry ; DIR_FCluster is the First Cluster of the 32 byte Directory Entry ;
DIR_Cluster	24	4	This cluster is interpreted based on the following DIR_LogType operations. DIR_LogType: Description File Write: Starting cluster in "Reserved FAT" of the cluster chain need to be appended to the file of this 32 byte directory entry. Starting from this cluster the cluster chain need to be copied to FAT1. File Truncate: Starting cluster of which the truncation should start File/Directory Delete, File/Directory Rename: The cluster value prior to DIR_Dcluster. The index value in the FAT which contains DIR_DCluster.
DIR_FileSize	28	4	File size in Bytes

TABLE II
Structure of Log Record with LFN in TI-LFAT File System

DIR_LogType	DIR_LFNCount	DIR_Name	…..	DIR_FileSize
1st 32 Bytes of Unicode LFN (16 Unicode Characters)				
2nd 32 Bytes of Unicode LFN (16 Unicode Characters)				
------ ------				
Nth 32 Bytes of Unicode LFN (16 Unicode Characters)				

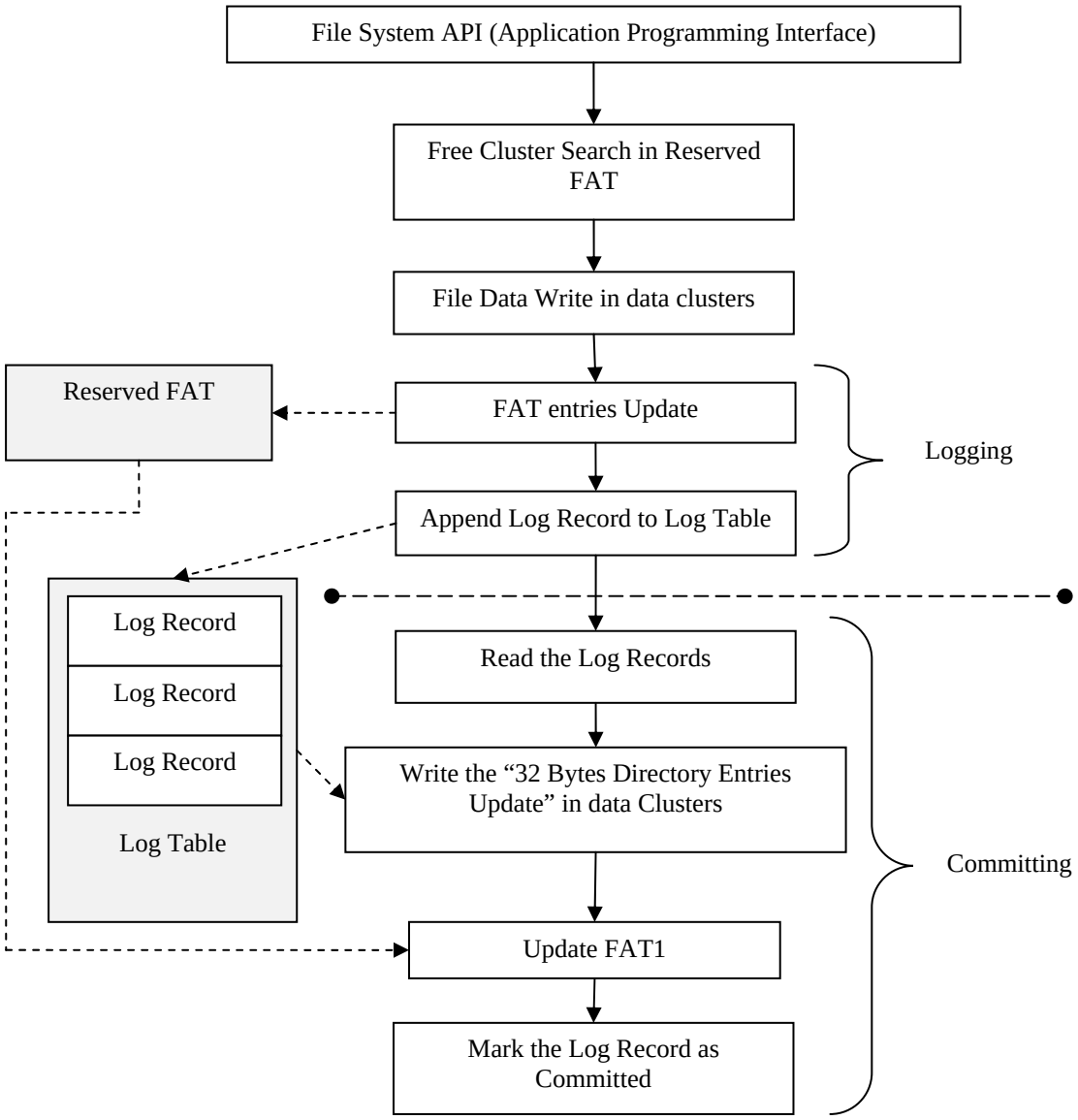

Fig.4. Flowchart of Logging and Committing

A. File/Directory Creation

Figure 5 shows the steps involved in the File/Directory creation with logging and committing operations. Note that, the FAT entry updates are required if there is directory creation. The Log records with file/directory name and allocated starting cluster number for directory are created and appended to the Log Table. The same log records are retrieved from Log Table and 32 bytes directory entries are created from the log records. These 32 bytes directory entries are written in to parent directory cluster. Here, parent directory means directory/folder

in which file/directory needs to be created. If the parent directory does not has enough free space to write the 32 bytes directory entries of the file/directory, then new cluster is allocated and appended to cluster chain of the parent directory. Allocating the new cluster to directory is referred as “Directory Update” operation. This Directory Update operation also requires logging and committing operations.

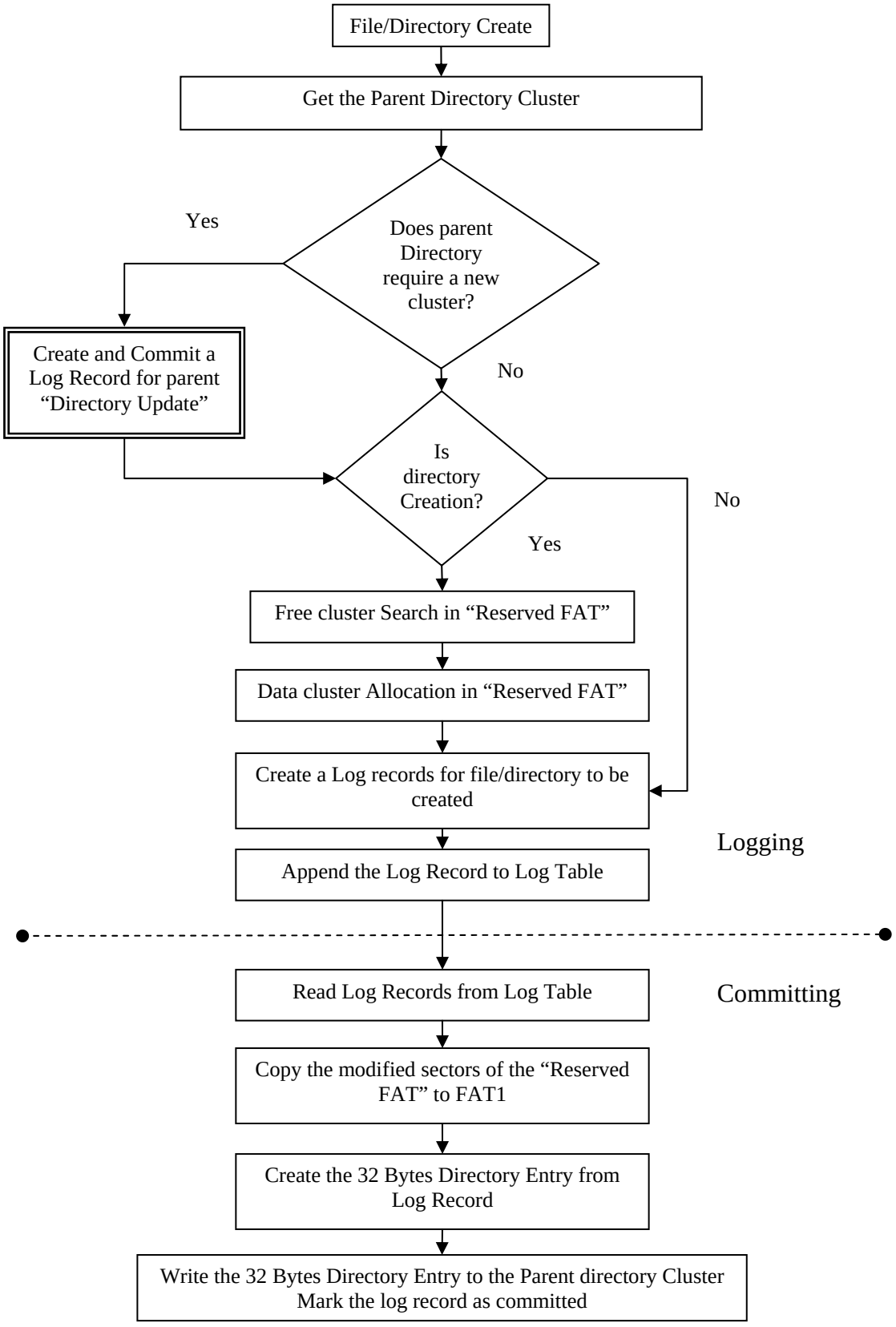

Fig.5. Flow chart of File/Directory Creation with Logging and committing

B. Directory Update

While a creating a file in a directory, if there is no sufficient space to create the 32 bytes directory entries of the file, then new cluster need to be appended to the cluster chain of the directory. The FAT entries update is involved in the operation and hence it requires logging and committing. The Log record with the DIR_LogType specified as "File Write", the Attribute indicating the "Directory" with the new cluster is appended to the Log Table. The cluster number is retrieved from the log record and appended to the cluster chain of the directory as part of commit operation. Figure 6 shows the flowchart of adding a new cluster to a directory with logging and commit operation.

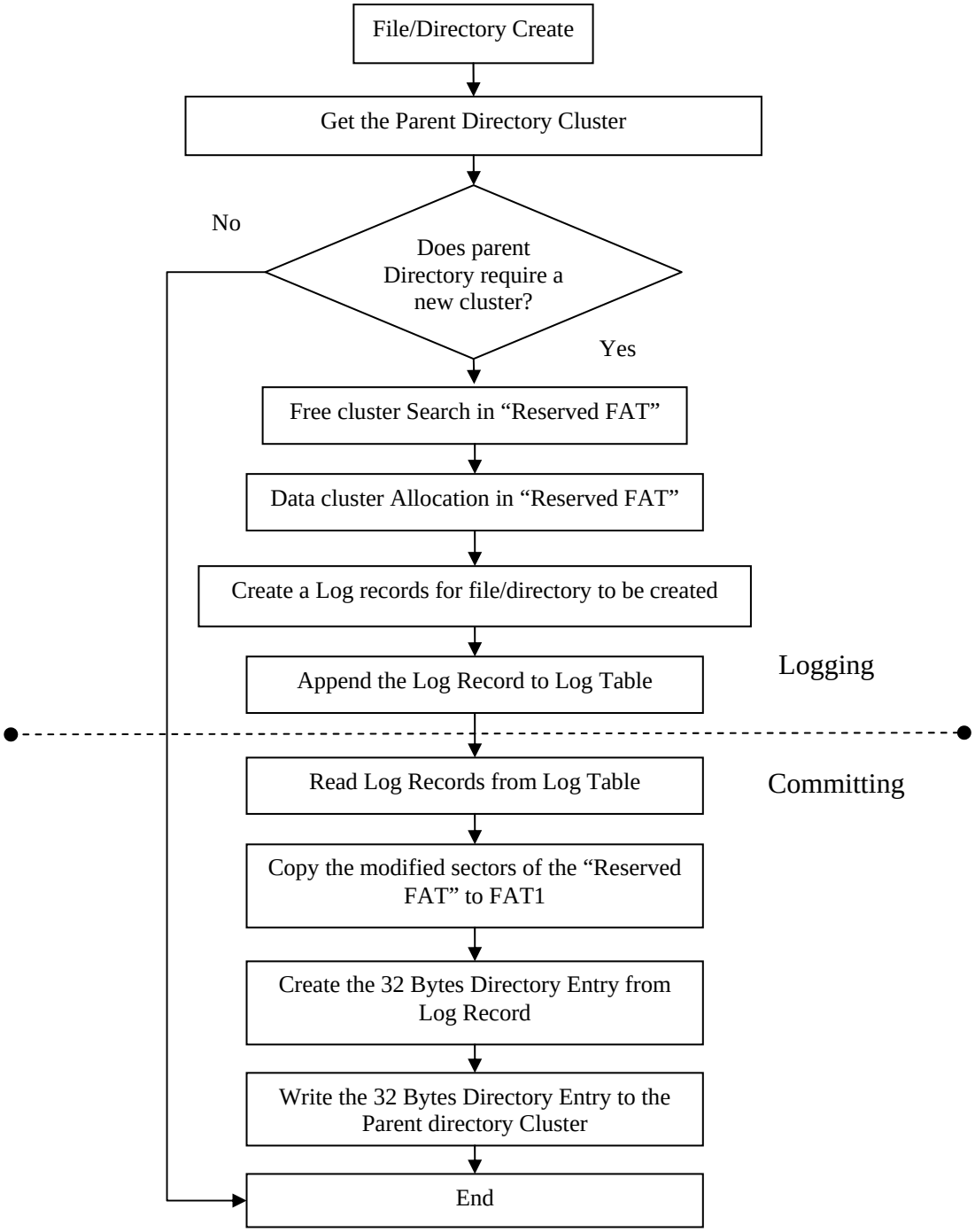

Fig.6. Flow chart of Directory Update with Logging and Committing

C. File / Directory Delete

The Algorithm for File/Directory Delete operation with logging and committing is as follows.

1) Create the Log record with DIR_LogType = File/Directory Delete
2) Get the first cluster of the cluster chain of the directory
3) Traverse the cluster chain and get all clusters of cluster chain of directory
4) If Deletion operation is on directory then
    a. Starting from last cluster, read the32 bytes directory entries of file/directory of the directory to be deleted.
    b. For each file or directory repeat this algorithm from step 2.
5) Else
    a. Free the clusters from last cluster to first cluster of cluster chain (reverse order) instead from first cluster to last cluster in Reserved FAT and FAT
6) Mark the log record as "committed"

This algorithm ensures that there are no orphan clusters if an uncontrolled power loss occurs while freeing the cluster chain in FAT during File/Directory deletion. In the example shown in figure 7 the clusters are free from cluster number 27 and 3 instead of 3 to cluster number 27.

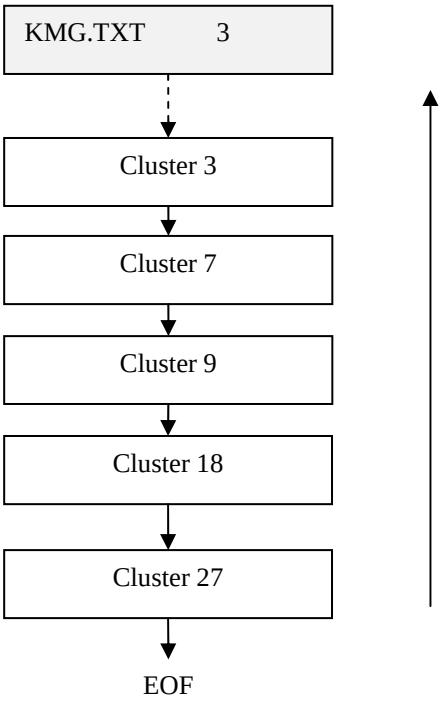

Fig. 7. Cluster chain deletion in reverse order during file/directory deletion

D. File Truncate

The File truncation is similar to File deletion operation. In case of file truncation, the log record with the DIR_LogType = truncate and DIR_Cluster is starting cluster which the truncation should begin. Starting from the last cluster, of the cluster chain of the file, to DIR_Cluster, the clusters freed in the reverse order. In the example, show in figure 8, the cluster 7 is the starting cluster from which the file deletion algorithm is applied to delete the clusters.

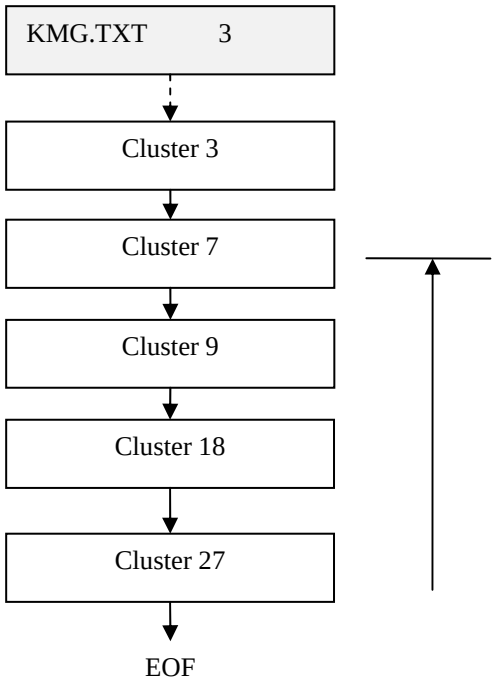

Fig. 8. Cluster chain deletion in reverse order during file truncation

E. File/Directory Rename

In case of file file/directory rename operation, minimum 2 log records are required. The first log record with DIR_LogType = Rename is created and it specifies the 32 bytes directory entry of the source file/directory. The second log record with DIR_LogType = "Sub log record" is created and specified the 32 bytes directory entry of the destination file/directory. If destination file/directory name if the LFN, the LFN log records are appended to the Log Table following the 2nd Log record. During the commit operation, 32 bytes directory entry of the file/directory specified in the 1st log record is marked as deleted and the new file/directory with name specified in 2nd log record and LFN log records is created in the at the DIR_DIndex of DIR_DCluster specified in the 2nd log record. The DIR_Fcluster specifies the cluster chain of the destination file/directory. Once the destination file/directory created, all the log records are marked committed by resetting the value of DIR_LogType all log records to 0. Table III shows the multi log structure for file/directory rename operation with LFNs.

TABLE III
Multi log structures for file/directory rename operation with LFNs

DIR_LogType = Rename	DIR_LFNCount	Source file/directory name	…..	DIR_FileSize
DIR_LogType = Sub Log Record	DIR_LFNCount = N	Destination file/directory name	…..	DIR_FileSize
1st 32 Bytes of Unicode LFN (16 Unicode Characters)				
Nth 32 Bytes of Unicode LFN (16 Unicode Characters)				

32 Bytes

F. File Write and Close

During file write operation new data of the file will be copied to newly allocated data clusters. The allocation of data clusters is first performed in the reserved FAT area as shown in figure 9. The DIR_LCluster and DIR_FCluster values are determined during the write operation. If there is power failure during the write operation, no file system update operation is performed in the next reboot and hence no changes are made to file.

During file close operation the log record with the DIR_LCluster and DIR_FCluster values and it is appended to the Log Table. The same log record is committed to the file system by copying the updated sectors of the Reserved FAT to FAT1 and updating the 32 bytes directory entry of the file as shown in figure 10. If there is power failure before appending the log record to the Log Table then no changes are made to the file. If there is a power failure after appending the log record, then in the next reboot, the log record is extracted and committed to the file system.

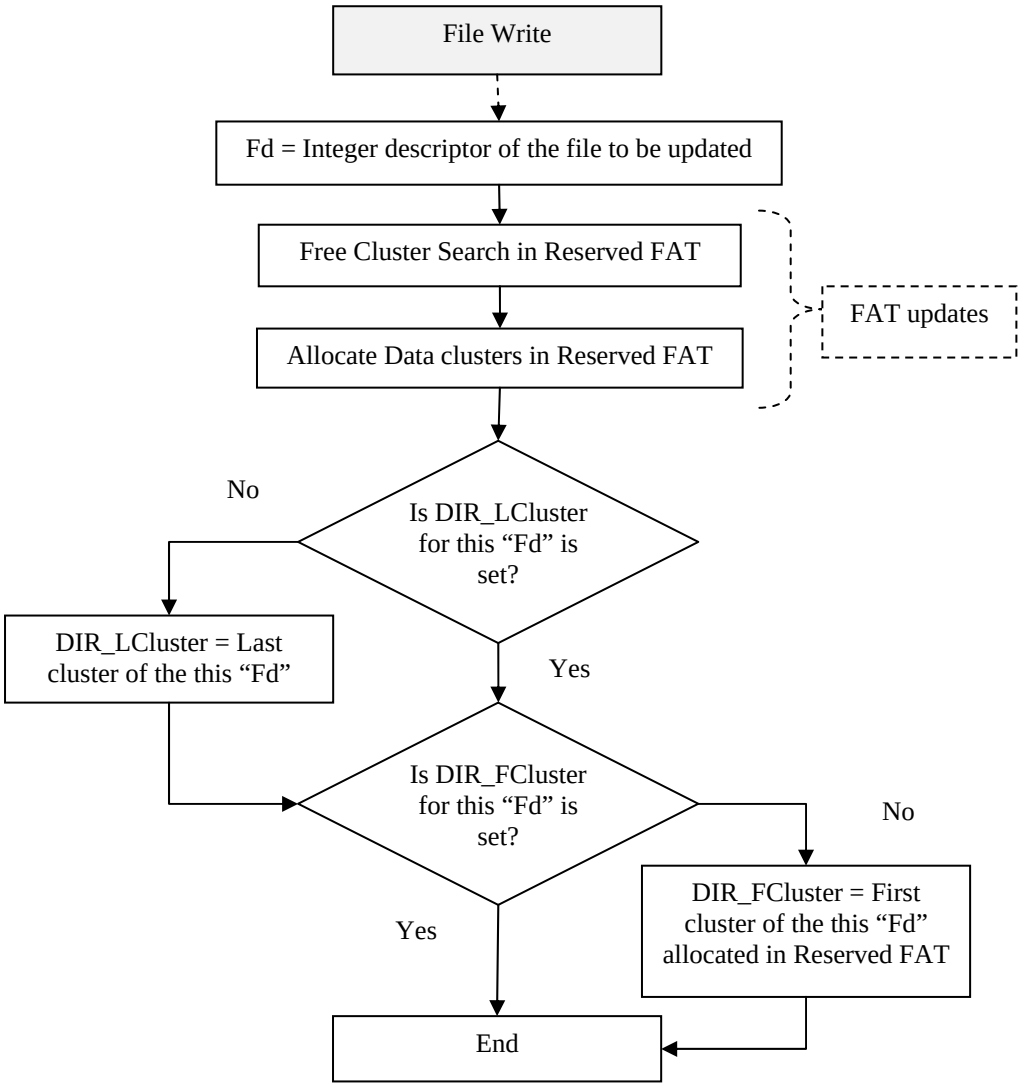

Fig. 9. File Write operation with FAT updates in Reserved FAT in TI-LFAT file system

IV. TI-LExFAT: Texas Instruments Log structured ExFAT file system

The structure ExFAT file system is similar to FAT file system, this file system is optimized for flash storage devices. Mainly, the ExFAT file system is differs with FAT file system as follows

1) Uses only single FAT (File Allocation Table) by default to improve the file system performance and life of the flash by reducing the meta data of the file system.
2) Cluster Heap [9] is used to optimize the free cluster search and allocation of clusters in case of contiguous clusters are available to allocate to a file. The file directory entry [9], stream extension directory entry [8] and file name directory entries [9] are used to represent the file/directory name, attributes, size, creation/update time and starting cluster of the file/directory. The FAT file system uses 32 bytes directory entry and LFN entries are used to represent the file/directory Meta data. The file directory entry, stream extension directory entry and file name directory entry are 32 bytes each. The size of the file/directory is stored by the field "data length" of the stream extension directory entry. This data length field is of 8 bytes whereas size field of 32 bytes directory entry of FAT file system is of 4 bytes. Hence, ExFAT file system supports higher file size than FAT file system.

3) The ExFAT file system supports the higher cluster size than FAT file system.
4) The ExFAT file system does not support SFN (Short File Names). The file name directory entry always stores the Unicode file/directory name.

The TI-LExFAT uses the same logging and committing techniques of TI-LFAT to support the power fail safe feature to ExFAT file system. Since, The ExFAT file system supports only one FAT instance, the "Reserved FAT" is placed as hidden file in root directory. The contiguous clusters are allocated the file containing "Reserved FAT". The Number of clusters required for the Reserved FAT is calculated using the "FatLength" [8] and "SectorsPerClusterShift" [8] values defined in the boot sector of the ExFAT file system. The simple calculation is as follows.

$$NumberofClustersof\text{"Re}servedFAT\text{"} \equiv K \equiv \frac{FATlength}{\left\lceil 2^{SectorPerClusterShit} \right\rceil}$$

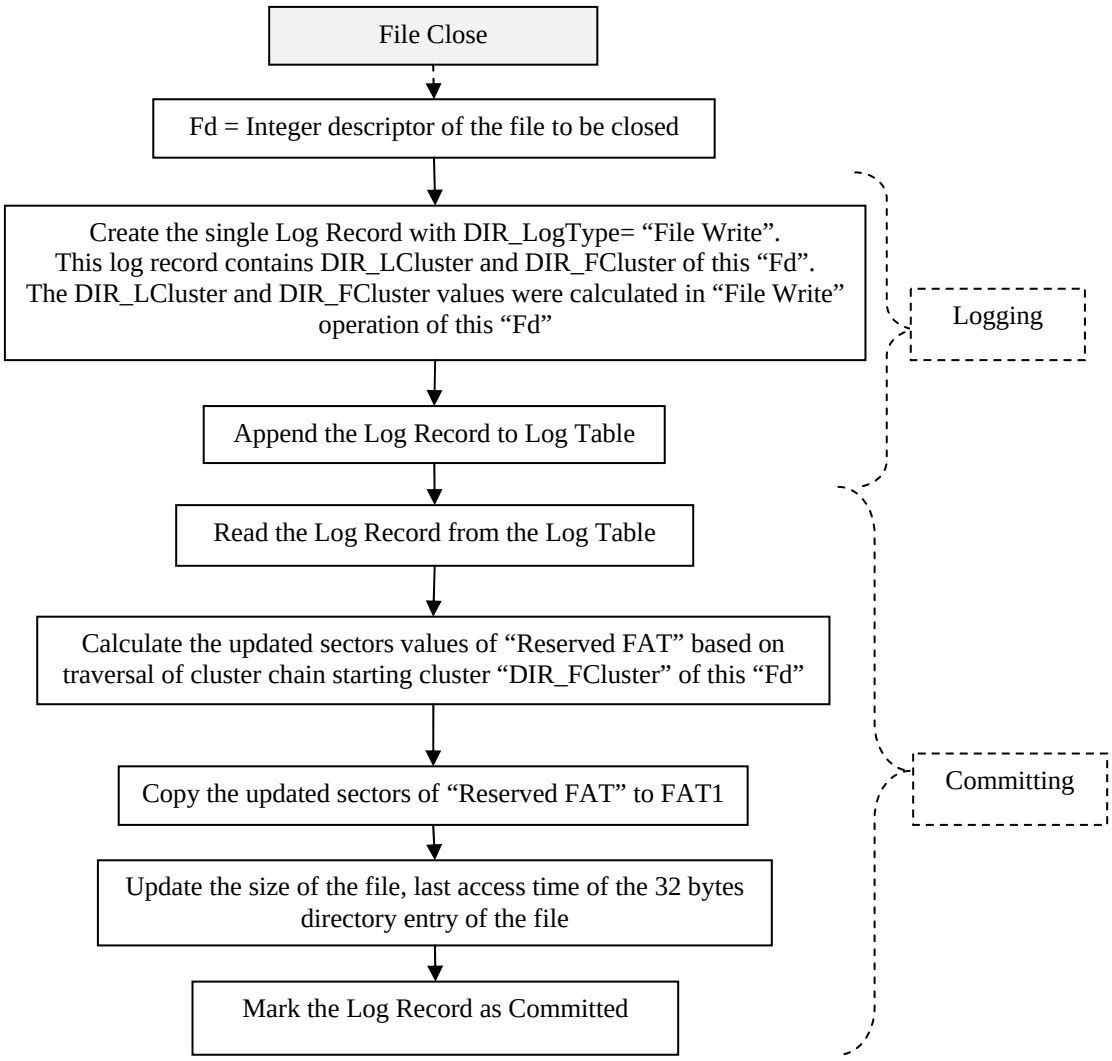

Fig. 10. File Close operation with 32 Bytes Directory Entry Logging and Committing in TI-LFAT file system

The ExFAT file system does not support the reserved clusters (status value 0xFFFFFFF6 in FAT), hence the TI-LExFAT file system can use hidden clusters by marking them as Bad clusters (status value 0xFFFFFFF7 in FAT) to store the Log Table. The Log table can also be placed as a hidden file in the root directory. The implementation of this paper, stores the Log table in as file in root directory. The Figure 11 shows the organization of TI-LExFAT file system with Reserved FAT is stored in the file RESERVE-FAT.LOG and the TI-LxFAT.LOG file contains the Log Table. Typically, the number of clusters allocated to the Log Table is less than the number of clusters allocated to the Reserved FAT. The ExFAT file system does not support SFN. The

size of file attribute is 2 bytes and file size is represented by 8 bytes. Hence, the TI-LExFAT file system modifies the log record structure of TI-LFAT as shown in Table IV.

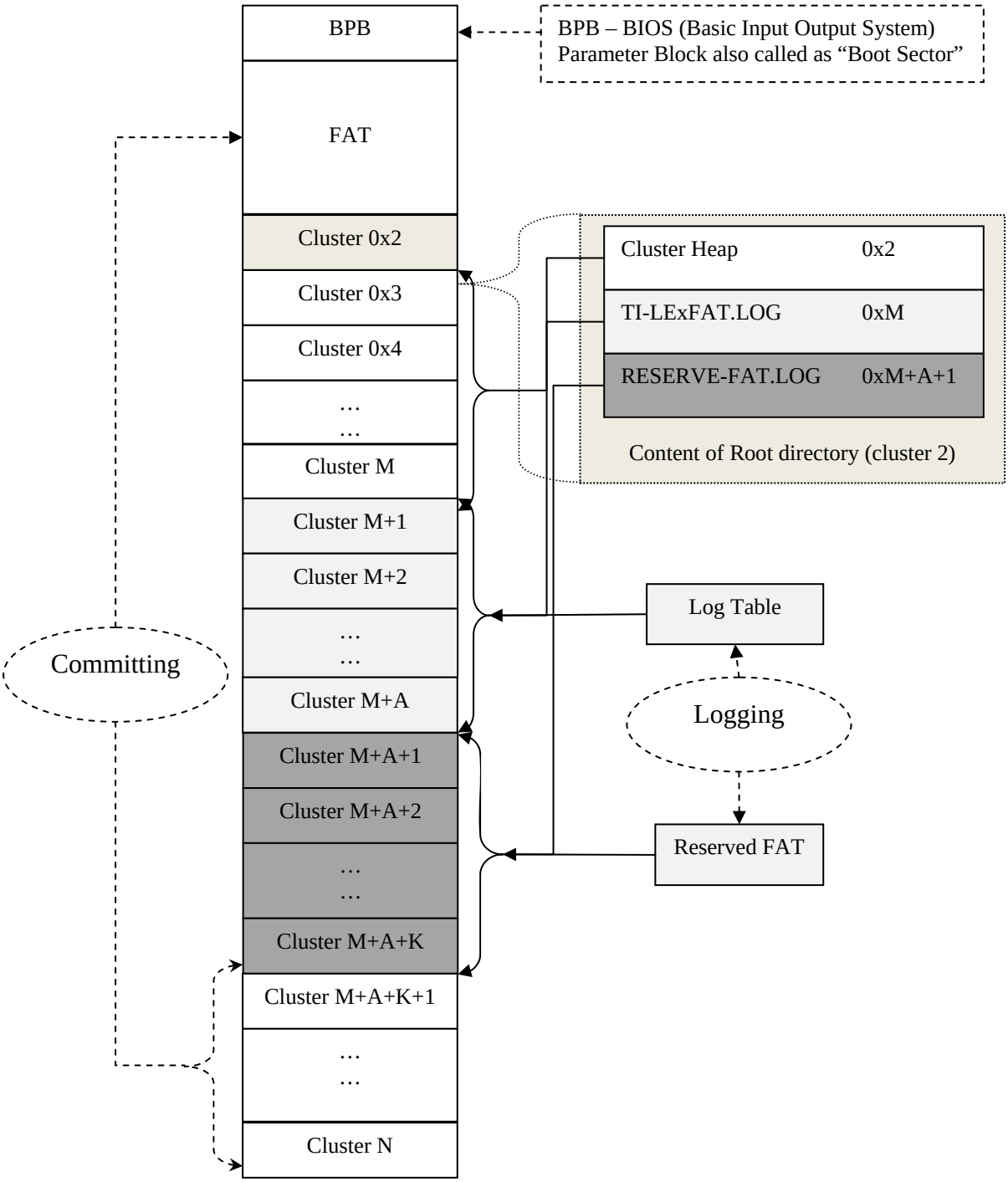

Fig.11. Organisation TI-LExFAT File System

Note that, the DIR_Hash field is used instead of SFN of the file/directory name. The DIR_Hash value contains the hash [8] value of file/directory name to be updated. While creating a file/directory this field contains the value 0. In TI-LExFAT file system, the file system operations such as file write, truncation and file/directory delete operations requires single log record whereas the file/directory create and rename operations requires multiple log records. During file /directory creation/rename operations the log records containing the Unicode file names are followed by mina log record with the DIR_LogType = "File/Directory Create" and main log record contains the DIR_Hash value instead of SFN. The TI-LExFAT files system uses the logging and committing operations of TI-LFAT file system to make the file system update operation as power fail safe. The

Logging and committing algorithms of the file/directory create, delete, rename, file truncation operations of TI-LFAT file system are applicable to TI-ExFAT file system. One additional activity in TI-LExFAT file system is during commit operation whenever the cluster status is copied from “Reserved FAT” to “FAT”, the cluster status is updated in the “Cluster Heap” also.

TABLE IV
Structure of 32 Bytes Log Record of TI-LExFAT File System

<table>
<tr><th>Name</th><th>Offset</th><th>Size in Bytes</th><th colspan="2">Description</th></tr>
<tr><td rowspan="10">DIR_LogType</td><td rowspan="10">0</td><td rowspan="10">1</td><td colspan="2">Type of the File system Operation</td></tr>
<tr><td>Value</td><td>File system Operation</td></tr>
<tr><td>1</td><td>File/Directory Create</td></tr>
<tr><td>2</td><td>File Write</td></tr>
<tr><td>3</td><td>File Truncate</td></tr>
<tr><td>4</td><td>File/Directory Delete</td></tr>
<tr><td>5</td><td>File/Directory Rename</td></tr>
<tr><td>6</td><td>Contiguous cluster chain allocated</td></tr>
<tr><td>7</td><td>File system operation of previous Log Record;
“Sub Log Record” and it is part of multiple log record to perform single file system operation.</td></tr>
<tr><td>0</td><td>Log Record is committed</td></tr>
<tr><td>DIR_LFNCount</td><td>1</td><td>1</td><td colspan="2">Length of Unicode file name</td></tr>
<tr><td>DIR_Hash</td><td>2</td><td>2</td><td colspan="2">Hash of the file name ; contains value 0 in case of file/directory creation operation</td></tr>
<tr><td>DIR_Reserv</td><td>4</td><td>4</td><td colspan="2">Reserved</td></tr>
<tr><td>DIR_Attr</td><td>8</td><td>2</td><td colspan="2">File/Directory Attributes</td></tr>
<tr><td>DIR_DIndex</td><td>10</td><td>2</td><td colspan="2">Index in the DIR_Dcluster; This index points to 32 bytes file directory Entry</td></tr>
<tr><td>DIR_DCluster</td><td>12</td><td>4</td><td colspan="2">Parent directory Cluster; The cluster in which this 32 bytes directory entry exists or to be created</td></tr>
<tr><td rowspan="6">DIR_FCluster / DIR_LCluster</td><td rowspan="6">16</td><td rowspan="6">4</td><td colspan="2">This value can contain the either first cluster or last cluster of the file/directory based on the DIR_LogType operation.</td></tr>
<tr><td>DIR_LogType</td><td>Description</td></tr>
<tr><td>File Write</td><td>Last Cluster of the stream extension Directory Entry;
DIR_LCluster is the last cluster of the cluster chain of the file before the write operation.</td></tr>
<tr><td>File/Directory Create</td><td rowspan="4">First Cluster of the Stream extension Directory Entry ;
DIR_FCluster is the First Cluster of the 32 byte Directory Entry;</td></tr>
<tr><td>File Truncate</td></tr>
<tr><td>File/Directory Delete</td></tr>
<tr><td>File/Directory Rename</td></tr>
<tr><td rowspan="5">DIR_Cluster</td><td rowspan="5">20</td><td rowspan="5">4</td><td colspan="2">This cluster is interpreted based on the following DIR_LogType operations.</td></tr>
<tr><td>DIR_LogType</td><td>Description</td></tr>
<tr><td>File Write</td><td>Starting cluster in “Reserved FAT” of the cluster chain need to be appended to the file of this 32 byte directory entry. Starting from this cluster the cluster chain need to be copied to FAT1.</td></tr>
<tr><td>File Truncate</td><td>Starting cluster of which the truncation should start</td></tr>
<tr><td>File/Directory Delete</td><td rowspan="2">The cluster value prior to DIR_Dcluster
The index value in the FAT which contains DIR_DCluster.</td></tr>
<tr><td>File/Directory Rename</td></tr>
<tr><td>DIR_FileSize</td><td>24</td><td>8</td><td colspan="2">File size in Bytes</td></tr>
</table>

A. File Write and Close in TI-LExFAT File System

During file write operation, In TI-LFAT file system performs the free cluster search and allocation of clusters is performed in "Reserved FAT" as shown in figure 9. The TI-LExFAT file system maintains the copy of Cluster Heap in the main memory for performance optimization. In TI-LExFAT file system, the free cluster search is always performed in the Cluster Heap, if the free clusters are in contiguous with previous allocated clusters of the file, then "FAT update" are not performed in Reserved FAT area as show in figure 12. The DIR_LCluster and DIR_FCluster values are determined during the write operation. If there is power failure during the write operation, no file system update operation is performed in the next reboot and hence no changes are made to file. During file close operation the log record with the DIR_LCluster and DIR_FCluster values and it is appended to the Log Table. While committing the log record, the cluster allocated to a file are contiguous starting from the cluster DIR_FCluster value then cluster allocation status is indicated only in Cluster Heap. If the free clusters are not in contiguous with previously allocated clusters of the file, the cluster status is indicated both in the Cluster Heap and Reserved FAT as show in figure 13.

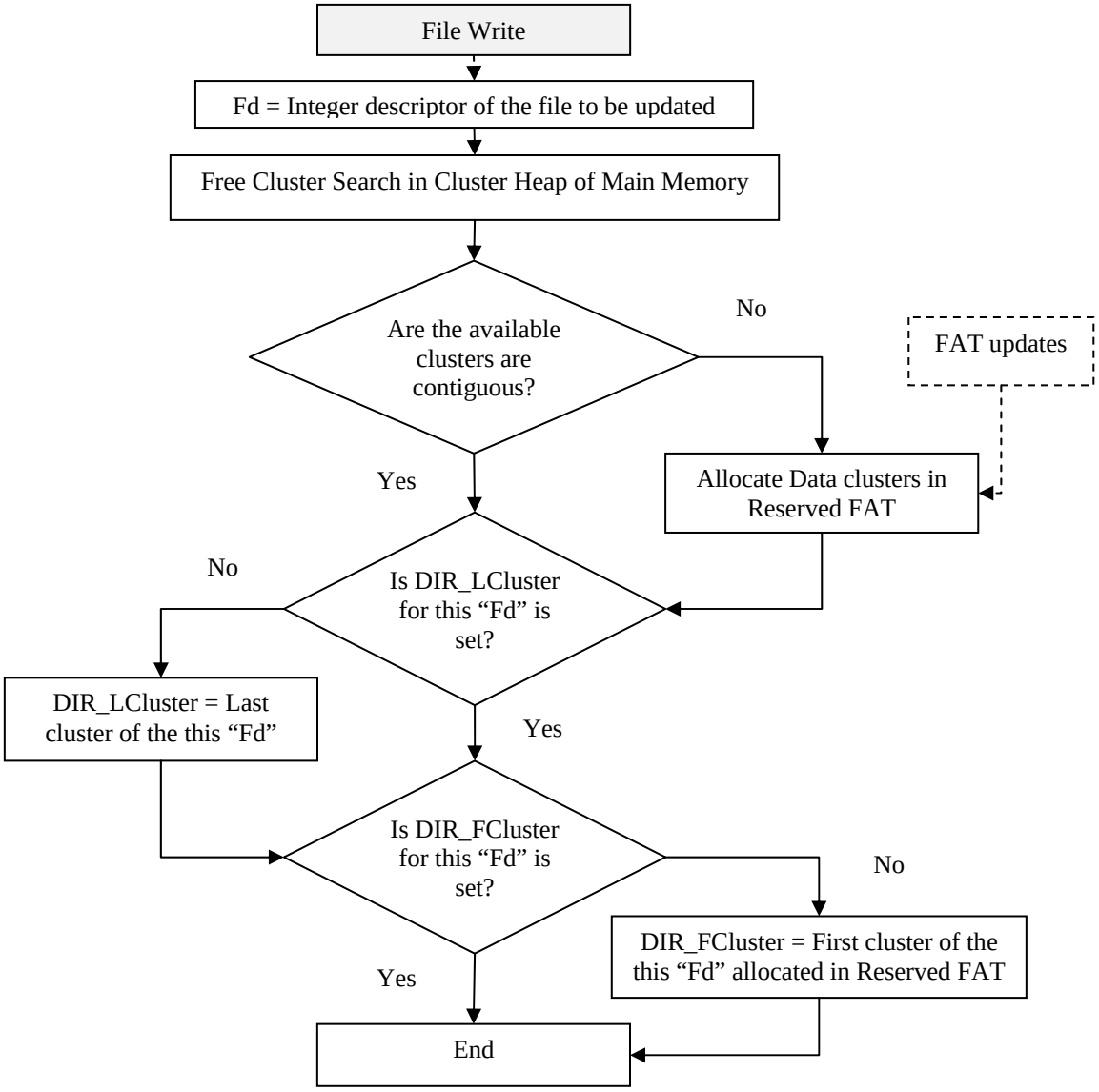

Fig. 12. File Write operation with FAT updates in Reserved FAT in TI-LExFAT file system

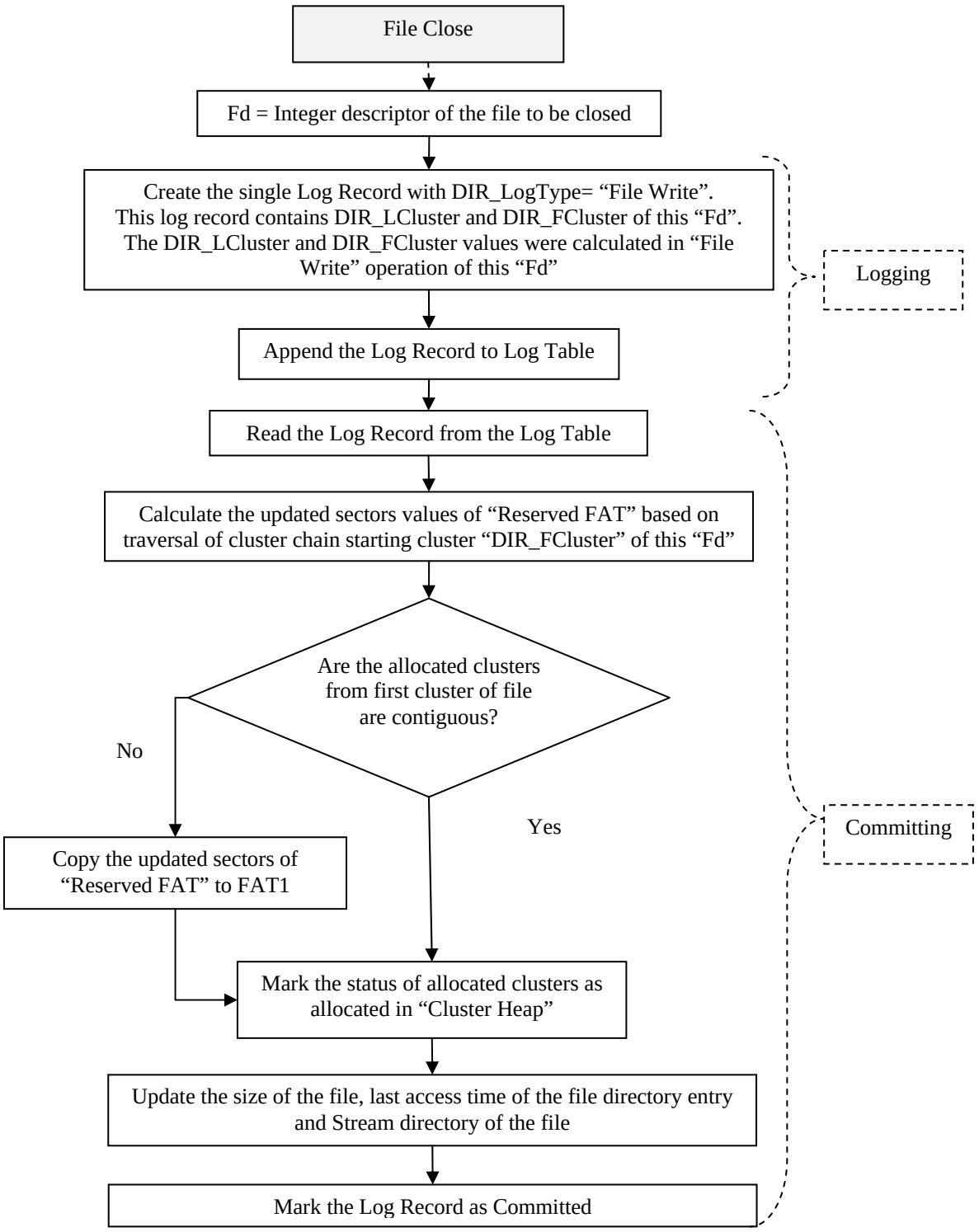

Fig. 13. File Close operation with log record creation and Committing in TI-LExFAT file system

V. EXPERIMENTAL RESULTS

The Logging and Committing operations for file write operation of TI-LFAT is implemented by modifying the existing FAT file system code of Linux kernel version 3.5.7 used in the Ubuntu operating system version 12.04. The Logging and committing operations for file write operation of TI-LExFAT is implemented by modifying the existing FUSE [34] based ExFAT file system code [35]. The Fuse based ExFAT file system is first modified as Linux kernel module and then same is extended to support the logging and committing operations of file write operation. The Performance Benchmarking of multiple file write operations are conducted on Dell laptop named Inspiron 5520 containing 4 Intel core i5 processors of 2.5 GHz speed and 4GB of RAM. The file system

benchmarking tool IOzone version 3.3 [36] [37] is used for the performance measurements. The IOzone software uses the record size of 4MB (Mega Bytes) for file read and writes operations. File sizes used for the performance benchmarking are 32MB to 512 MB. The SanDisk 64GB SD card is used for the performance benchmarking and it is mounted with buffer cache disabled in the Linux kernel. The Figure 14 shows the file write performance of FAT32, TI-LFAT and ExFAT and TI-LExFAT file systems. The average performance of FAT, TI-LFAT, ExFAT and TI-LExFAT file systems are 2.7 MB/S (Mega Bytes per Second), 2.57MB/S , 7.4 MB/S and 5.79 MB/s respectively. The FAT file system always uses the two instances of File Allocation Tables. The Logging operation causes the additional files system write and it cause reduction in file write performance. The TI-LFAT uses one FAT as the "Reserved FAT" for "FAT updates" and hence the reduction in the performance of TI-LFAT is minimal in comparison with FAT32 file system. The file write operation in TI-LFAT file system yields performance degradation of 4% to 9% in comparison with FAT32 file system. The ExFAT file system uses only one FAT by default. The TI-LExFAT includes an additional FAT as "Reserved FAT" hence the performance reduction is 18% to 34% compared to ExFAT file system. If the contiguous clusters are allocated to file during write operation then reduction in performance is minimal.

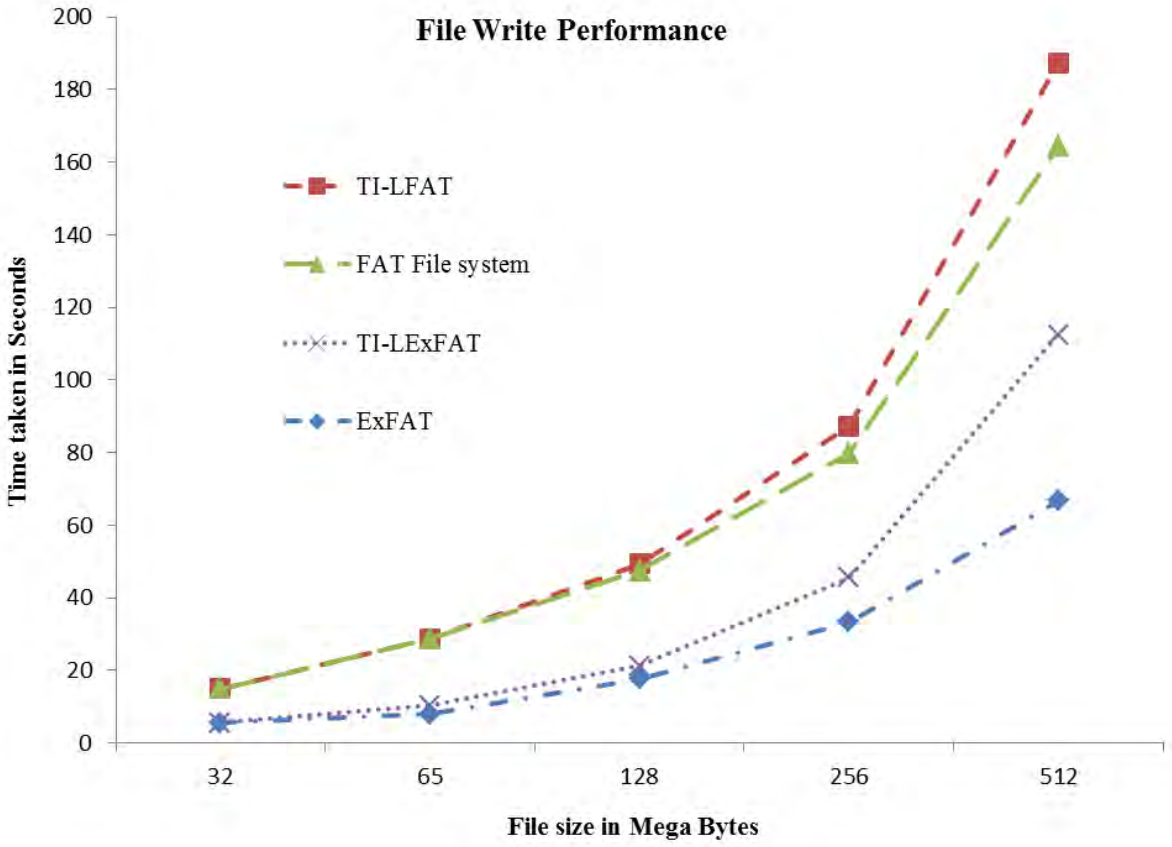

Fig. 14. File Write performance comparison between FAT32, TI-LFAT, ExFAT and TI-LExFAT File systems

VI. CONCLUSION

The complete design and implementation details of the TI-LFAT and TI-LExFAT file systems are discussed in this paper. The TI-LFAT takes the advantages of both KFAT and TFAT file systems. If there is an abrupt power failure or sudden removal of storage device from the computer system during the "FAT update" operation in Reserved FAT can cause the orphan clusters in the Reserved FAT of TI-LFAT file system. The orphan clusters can be made as free clusters by copying the entire FAT content to Reserved FAT. During file/directory creation and file write operation, if there is no free cluster in Reserved FAT, then the FAT can be copied to Reserved FAT to convert the orphan clusters to free cluster. The TI-LFAT uses the minimal log records compare to KFAT file system. Hence, loss of performance is minimal during file write operation in TI-LFAT file system. The Logging and Committing operations of TI-LFAT file system are applicable to TI-LExFAT file system. The usage of Cluster Heap in the TI-LExFAT avoids the creation of orphan clusters during uncontrolled power failures or sudden removal storage device from the computer system. The cluster heap avoids the FAT updates in "Reserved FAT" if the contiguous clusters are allocated to a file and hence minimizes the performance degradation.

ACKNOWLEDGMENT

It is a pleasure to acknowledge Mr. Robert Shullich, Enterprise Security Architect, Tower Group Companies, USA, for resolving technical queries to understand the ExFAT file system. Mr. Andrew Nayenko deserves special thanks for open souring the FUSE based ExFAT file system implementation to Ubuntu and other Linux operating system distributions.

REFERENCES

[1] Microsoft, *"FAT32 File System Specification, FAT: General Overview of On-Disk Format"*, 2000.
[2] *Multimedia card specification version 4.4*, JEDEC (Joint Electron Devices Engineering Council) standard, 2008.
[3] *SD Specifications Part 1: Physical Layer Simplified Specification version 4.10*, SD card Association, January 22, 2013
[4] Supriya Kulkarni P, Jisha P, "Study of Bad block management and Wear leveling in NAND flash memories", International Journal of Research in Engineering and Technology (IJRET), Volume 2, Issue 10, October 2013.
[5] Intel Corporation. Understanding the flash translation layer (FTL) specification, December 1998.
[6] New Technology File System (NTFS) , http://www.ntfs.com
[7] Rajeev Nagar, Windows NT File System Internals: A Developer's Guide, Orelly publications, September 1997.
[8] Ravishankar V.Puipeddi, Vishal V.Ghotge, Ravinder S.Thind, "Quick file name look up using name hash", USPTO Patent: 8,606,830, granted on November 20, 2013.
[9] Keshava Munegowda, Venkatraman S, Dr. G T Raju, "The Extended FAT file system: Differentiating with FAT32 file system", Linux Conference, Prague, Czech Republic, Europe, October 2011.
[10] Keshava Munegowda, "Power Fail-Safe FAT file system", Embedded Linux Conference, San Francisco, CA, USA, April 2011.
[11] Mendel Rosenblum and John K. Ousterhout. "The Design and Implementation of a Log-Structured File System", Proceedings of the 13th ACM Symposium on Operating Systems Principles and the February 1992 ACM Transactions on Computer Systems.
[12] Mendel Rosenblum "The Design and Implementation of a Log-Structured File System", PhD Thesis, the Springer International Series in Engineering and Computer Science, ISBN: 0792395417, December 1994.
[13] Remy Card, Theodore Ts'o, Stephen Tweedie, "Design and implementation of the Second Extended File system", First Dutch International Symposium on Linux,
[14] Dr. Stephen C. Tweedie , "Ext3 , journaling file system", Ottawa Linux Symposium, Ottawa Congress Centre, Ottawa, Ontario, Canada on the 20th of July, 2000.
[15] Mathur, Avantika; Cao, MingMing; Bhattacharya, Suparna; Dilger, Andreas; Tomas, Alex; Vivier, Laurent (2007)."The new ext4 Filesystem: current status and future plans". Proceedings of the Linux Symposium. Ottawa, ON, CA: Red Hat. Retrieved 2008-01-15.
[16] Btrfs file system, https://btrfs.wiki.kernel.org/index.php/Main_Page
[17] David Woodhouse, JFFS: The Journaling Flash File System, Ottawa Linux Symposium 2001.
[18] J¨orn Engel, Robert Mertens, "LogFS - finally a scalable flash file system", 2008.
[19] UBIFS file system, http://www.linux-mtd.infradead.org/doc/ubifs.html.
[20] YAFFS, A Nand Flash File system, Embedded Linux conference, Europe -2007.
[21] Joo-Young Hwang, "F2FS: A New File System Designed for Flash Storage in Mobile", Embedded Linux conference Europe, Barcelona, Spain, November 2012.
[22] M.S. Kwon, S.H. Bae, S.S. Jung, D.Y. Seo, and C.K. Kim, "KFAT: Log-based transactional fat file system for embedded mobile systems", in proceedings of the US-Korea Conference on Science, Technology, and Entrepreneurship. 2005.
[23] Samsung, "Linux RFS Power off Recovery", July 2008.
[24] Samsung, "TFS4 Power off Recovery", August 2007.
[25] Gui-Jung Lee, Jung-Gi Kim, "Device and Method for Data Recovery in File System", USPTO Patent: 5974426, granted on October 26, 1999.
[26] Brain C Kuschak, "File system for avoiding loss of data", U S patent: 6742079, granted on May 25, 2004.
[27] Chih-Chuan Tag, Hung-Lin Chou, "Journaling FAT file system and access method thereof", US patent: 7979804, granted on Jun 28, 2011.
[28] Nam Ho Kim, Yun Seop Yu, "HFAT: Log-Based FAT file system using Dynamic Allocation Method", Journal of information and communication convergence engineering (JICCE), December 2012.
[29] Keshava Munegowda, Veera Manikandan Raju, Madan Srinivas, Rohit Joshi, "FAT file in Reserved Cluster with Ready Entry State", US patent : 8452734, granted on April 19, 2013.
[30] Michael D. Malueg, Hang Li, Gopalan, ORadko, Polivy, Drasnin, Farmer, Huang, "Transaction Safe FAT file system", US patent :8156165, April 10, 2012.
[31] TexFAT Overview (Windows Embedded CE 6.0), http://msdn.microsoft.com/en-us/library/ee490643(v=winembedded.60).aspx
[32] Liang Alei, Liu Kejia, Li Xiaoyong, Guan Haibing, "FATTY: A Reliable FAT file system", 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2007, IEEE Computer Society.
[33] Keshava Munegowda, Dr. G T Raju and Veera Manikandan Raju, "Adapting Endurance and Performance Optimization Strategies of ExFAT file system to FAT file system for embedded storage devices", International journal of Engineering and Technology (IJET). Volume 6, Issue1, ISSN: 0975-4024, February 2014.
[34] FUSE (File System in User Space) : http://fuse.sourceforge.net
[35] FUSE based ExFAT implementation for Linux : http://code.google.com/p/exfat/
[36] Keshava Munegowda, Dr. G T Raju, Sourav Poddar, "FFSB and IOzone: File system Benchmarking Tools, Features and Internals", Embedded Linux Conference, Barcelona, Spain, Europe, November 2012.
[37] IOzone, "File systems Benchmarking tool", http://www.iozone.org