Fault Tolerant File Systems

Mayank Singh Chauhan 2016CS50394
Mankaran Singh 2016CS50391
Atishya Jain 2016CS50393
Avaljot Singh 2016CS50389

Group 1

August 6, 2019

1 Introduction

Fault-tolerant[1] technology is a capability of a computer system, electronic system or network to deliver uninterrupted service, despite one or more of its components failing. Fault tolerance also resolves potential service interruptions related to software or logic errors. The purpose is to prevent catastrophic failure that could result from a single point of failure.
Such systems are good at detecting a failure of CPU, I/O subsystem, memory cards, motherboard, power supply or network components. The failure point is identified, and a backup component or procedure immediately takes its place with no loss of service.
In this report, we discuss the faults due to *block read error* in the FAT file system. We also propose a file system in which the loss of information is minimized and is ideally proportional to the number of blocks which are unreadable. Lastly, we discuss a strategy for a completely fault tolerant file system.

2 FAT and associated limitations

Originally designed in 1977 for use on floppy disks, FAT[2] was soon adapted and used almost universally on hard disks throughout the DOS and Windows 9x eras for two decades. Today, FAT file systems are still commonly found on USB sticks, flash and other solid-state memory cards and modules, and many portable and embedded devices.

2.1 What is FAT?

FAT file system operations can be explained in the following points:

1. FAT table consists of various entries, each of which has 3 things. **Metadata, Pointer to the block in Disk, Pointer to the next entry in FAT table**.

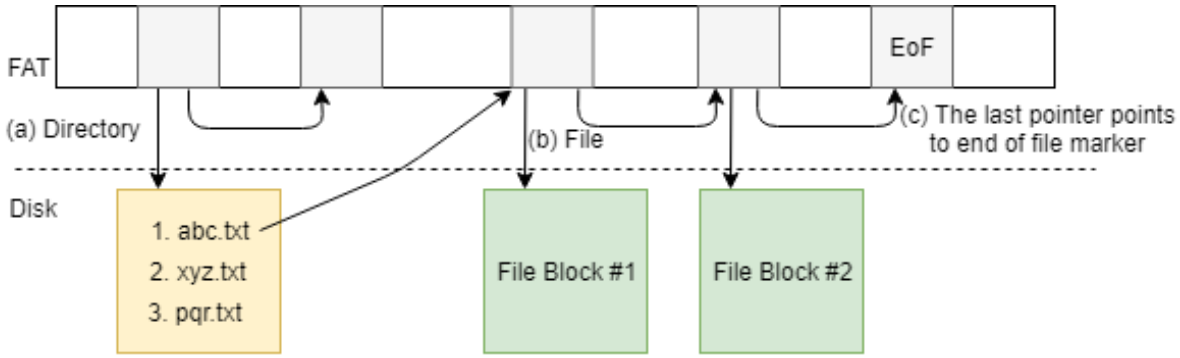

Figure 1: File Allocation Table (FAT)

2. If a FAT table entry corresponds to a directory, the pointer to disk points to a block which comprises of metadata and pointers to FAT table entries of daughter files and folders as in *(a)* above. The contents of the directory may not be stored as a single block on the disk but rather split across multiple blocks and hence, the FAT table entry of previous block points to the entry of the next block in FAT.

3. If a FAT table entry corresponds to a file, FAT contains the pointer to the disk which stores the actual file data as a block. Just like in directories, files may be split across blocks and the previous file table entry contains the pointer to the next file table entry which stores pointer to the second block of data as in *(b)*.

2.2 Types of possible block read errors

As described above, there exist three types of references:- FAT to disk, FAT to FAT, disk to FAT. A file system of this type can be seriously damaged by a block read error as follows:

1. **Boot Sector Error**: Boot sector in FAT file system exists at a fixed partition on the disk which is usually the very first block on the disk. If this block gets corrupted, the OS won't boot at the first place.

2. **FAT Table entry read error**: FAT table entries serve as the main lookup table for each file or directory in the file system, but initially the whole of FAT table is also loaded into the memory from the disk when system performs booting up operations. If the block corresponding to a FAT table entry itself gets corrupted, the whole of information pointed to by that block gets lost. This information is not proportional to the lost as if its a directory, the pointers and information to all its sub-directories and files are lost. If its a file, all the subsequent file entries get lost as the pointer pointing to them gets corrupted.

3. **Directory block read error**: This will cause the disk to FAT pointers for daughter files/directories of the corrupted directory to be lost and correspondingly all the data pertaining to them as there is no other way to reach the files.

4. **File block read error**: This corruption won't cause disproportionate loss of data but only the part of the file residing on that specific block gets lost. All the pointers are intact as they reside in the FAT table itself.

Thus a block read error is really fatal to this file system. Let's see how we can improve this in the following sections.

3 File System with Loss proportional to number of Unreadable Blocks

The main reason why there is a huge data loss when a block gets corrupted is that the FAT pointer to daughter files/directories also got lost and then it becomes impossible to recover them. To tackle this problem, we can store the next FAT table pointer on the FAT table itself. This way, we will be able to access the daughter files/directories even if the parent directory gets corrupted. This way, whenever a block gets corrupted, we will only lose data corresponding to that block.
The above change will resolve the directory block read error but we still need to tackle the Boot sector error and the FAT table entry read error. For that we will maintain a copy of boot sector which will be used when the original boot sector gets corrupted. Also, in the original FAT file system, a copy of FAT table always exist in the disk. So we don't need to worry about the corruption of the disk block containing the FAT table.
The following figure shows the modified File System:

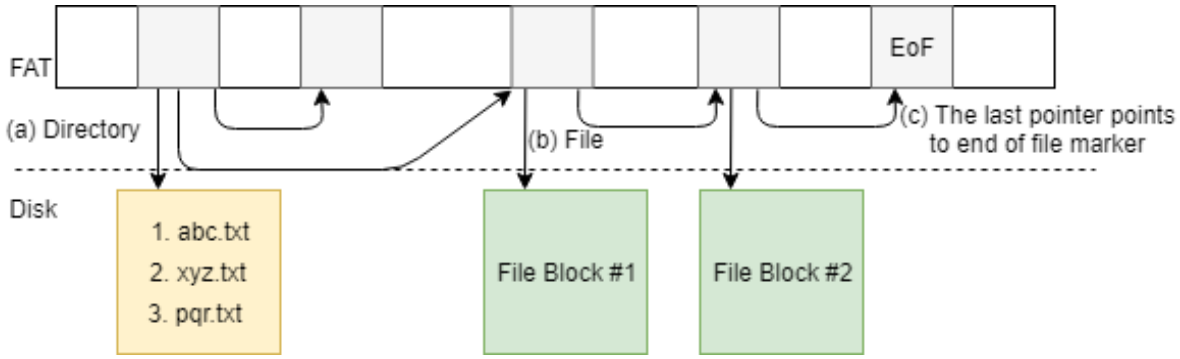

Figure 2: Modified FAT

3.1 Advantages of the new File System:

1. **Boot Sector Error and FAT table entry read Error**: Both these errors are resolved since we have a copy of both the Boot Sector as well as the FAT table in our new file system.

2. **Directory block read error**: In our new File system, since the FAT pointers for daughter files/directories of a directory are stored in the FAT table itself, even if the

parent directory gets corrupted, we will still be able to recover the daughter files/directories. This makes the loss of data proportional to the number of sectors corrupted.

3. **Fast Access**: Since the FAT pointers to the daughter files/directories are there in the FAT table (present in the memory), their overall access time becomes much less as compared to the time taken in case of original FAT file system.

3.2 Disadvantages of the new File System:

1. **FAT table becomes very large**: Since we are storing the daughter files/directories pointers in the FAT table itself, the size of FAT table can get extremely large. In such a case, most of the main memory will be occupied by the FAT table itself degrading the performance of other processes.

2. **Loss of Metadata**: A parent directory also stores the metadata of the daughter files/directories in addition to the FAT pointers. So storing only FAT pointers may not be sufficient to completely recover the daughter files/directories. If we stored the essential part of the metadata also in the FAT table, it might result into an extremely large FAT table which will degrade the performance of the system.

3. **Booting becomes slower**: Due to larger size of FAT table, fetching FAT table from disk to memory becomes slower. Since this occurs at boot time, booting becomes slower.

4. **Memory wastage due to presence of copies**: Since we are maintaining a copy of boot sector and FAT table, we are wasting space as these copies will not be used until and unless the blocks containing the original versions got corrupted.

4 Completely fault tolerant file system

Here we will propose a design with certain assumptions in mind. We will incrementally proceed towards a completely Fault tolerant system

4.1 Maximum N blocks Read Error simultaneously

4.1.1 Assumption

For this design we make an assumption that when a sector on a disk gets corrupted, a maximum of 1 block can face corruption.

4.1.2 Description

We keep a ***check*** block on the disk, which essentially contains XOR of all the bits of all the blocks. In other words, kth bit of parity block is XOR of kth bits of all the blocks. So, in case if the parity block gets lost, it can be easily regenerated. If any other block on the disk is lost, it can be recovered by taking XOR of all the remaining blocks and the parity block.

The same concept can be extended to N blocks case, using the concept of *Haming codes*[3] used for error detection and recovery in networks, where we need to have $1 + \lceil log_2 N \rceil$ check blocks for a fault tolerance over N blocks.

4.1.3 Pros

1. We do not replicate the data but just store some additional blocks of total data
2. The XOR operation can be done extremely fast in hardwares.

4.1.4 Cons

1. We can take care of only $N/2$ corrupt blocks at a single time. Its fault tolerance increases with the increase in value of N
2. Adds an overhead of taking XOR on every edit of the files.

4.2 File System: Replicating Data

4.2.1 Assumption

We assume here that whenever a disk sector gets corrupt, only continuous chunks of sectors get corrupted. It is never the case that two completely different parts of disks get corrupted simultaneously, which is a fairly valid assumptions because most of the times a particular block in our disk gets damaged and we lose access to everything because of that.

4.2.2 Description

In order to design completely fault tolerant file system, we realize that we can't do that without duplicating the data. So, all we can do is to replicate all the data and store it optimally and handle it efficiently so that we cut down all the extra time and space costs incurred. Since we make a copy of all the files, we also need to maintain the pointers to these duplicate file locations separately. So, we also define another FAT table and name it *dupFAT*. This new data structure is essentially similar to our old FAT table. The only difference is that it stores the links and addresses of the duplicate files and directories (if it all treated separately).
Concretely, we need to do the following:

- Whenever a new file is made, make two copies of the file at two very different locations in the disk. This will ensure the safety of one of the copies even if one sector/block of the disk crashes.
- Update the FAT and dupFAT entries appropriately.
- Whenever, an existing file is accessed, it is looked up in the FAT table. If the corresponding block of the disk crashes, we lookup for its duplicate in the dupFAT table. The only similarity that can be used here for the additional lookup is that the absolute path of the file will be same in both the file systems.

- Additional care now needs to be taken whenever an existing file is edited or deleted. The same operation needs to be done for the copy as well.

4.2.3 Pros

- A very trivial advantage of this solution is that we will not lose any data until only one of the copies of any file existed in the block of the disk that crashed. This solution can however be extended to making N copies of each file as implemented in MooseFS[4]. In such an implementation, we can recover a file even if N-1 copies of the file get lost.

4.2.4 Cons

- The major loss that we incur in this solution is that we need more space to duplicate the data.
- The time complexity of every file operation nearly doubles.

5 Conclusion

We have analysed the FAT file system, looked at its possible failures and designed the file systems to overcome those failures. Loss proportionate File system had an advantage of proportionate loss without replicating any data, but has a disadvantage of blown-up FAT table size. Completely Fault tolerant file system proposals follow an assumption of continuous losses and involve some data duplication but have an advantage of no loss in any data. Thus, the decision of best file system depends on the nature of requirements and resources available for the system.

References

[1] https://en.wikipedia.org/wiki/Fault_tolerance

[2] https://en.wikipedia.org/wiki/File_Allocation_Table

[3] https://en.wikipedia.org/wiki/Hamming_code

[4] https://en.wikipedia.org/wiki/Moose_File_System