Oriental Journal of Computer Science & Technology Vol. **3**(2), 269-271 (2010)

Some notable reliability techniques for disk file systems

WASIM AHMAD BHAT and S.M.K. QUADRI

Department of Computer Sciences, University of Kashmir, Srinagar (India).

(Received: November 06, 2010; Accepted: December 10, 2010)

ABSTRACT

File system operations include data operations and metadata operations. Data operations act upon actual user data while metadata operations modify the structure of the file system, like creating, deleting, or renaming files, directories, etc. During a metadata operation, the system must ensure that data are written to disk in such a way that the file system can be recovered to a consistent state after a system crash. In this paper we look at some most notable techniques which ensure reliability of disk file systems against system crashes and failures.

Key words: Disk file system, reliability techniques, data operations.

INTRODUCTION

File system metadata includes directory organization, file size, block spanning information, resource allocation, etc. This metadata controls the structure of a file system and hence maintaining the integrity of metadata is the key of file system reliability. Depending upon the type of file system, metadata is located in several different places and hence file operations that require updating file system metadata need to update data located in these different places, e.g. a file append operation may require three metadata updates: 1) finding an unused data block, and marking it as allocated, 2) adding it to the file's list of clusters; 3) rewriting the size of the file. Ideally, all these updates should be done atomically. However, if a crash or failure happens in between, the file system is left in an inconsistent state. It is not possible to make sure absolutely no data is lost after a system crash[1]. However, many studies have managed to keep the structure of various file systems consistent across crashes and failures. In this paper we will look at some most notable approaches that ensure disk file system reliability.

Offline Check

Many traditional disk file systems, such as FAT[2] and ext2[3] do not care during file operations to keep their structure consistent across crashes. When a crash occurs, the system does an offline file system consistency check to identify the inconsistent structure and possibly rectify it. The FSCK[4] program scans the whole disk and fixes any inconsistencies it finds. This approach works fine for small disk volumes, but as file system volumes are getting larger and larger, the offline time spent on FSCK becomes intolerable.

Synchronous Write

An in-memory buffer cache is used by most of the file systems to improve the performance. In these file systems, when some data needs to be written, it is first cached in memory, and written back asynchronously at some later proper time. Because the order in which the buffers in the cache are written back is not guaranteed, various reliability issues could arise when a crash occurs. For example, if a crash happens when the system is doing a file append operation, it is possible that an unused cluster is added to one file's cluster list, while it is not marked as allocated. If the system restarts and resumes work without noticing this, it is likely to allocate the same cluster again to another file, causing a joint-cluster inconsistency problem. The original BSD Fast File System (FFS)[5] has used synchronous write to properly sequence related metadata updates. For example, it is assured that

the cluster is marked allocated before it is added to the file's cluster list when doing a file's appending operation. In this way, even if a crash occurs between these operations, only one cluster is lost, no further corruption of the file system will be induced, thus eliminating the compulsory offline FSCK on the next boot.

Soft Updates

In Synchronous Write all metadata updates are done synchronously, as such file system operations like file creation and deletion are forced to proceed at disk speed rather than processor or memory speed. Since disk IO speed is much slower than memory, synchronous write reduces system performance and is its main drawback. Soft updates[6] provide a way to eliminate most synchronous writes in the FFS, while still keeps the same level of reliability. The soft updates approach maintains the dependencies of updated metadata buffers in memory, and writes them back based on these dependencies. Because not all groups of metadata updates have dependencies between each other, some of these groups may be merged and reorganized, thus eliminating many unnecessary disk writes. The main drawback of soft updates is that it is too complex to understand and implement.

Logging

Another popular method of improving file system reliability is using file system log. In log-based file systems, such as ext3[7], all metadata updates of a file operation are first written to the log and grouped as a transaction, and then committed to blocks where metadata is located. When a crash occurs, the system first checks the file system log, redoing the completed but not committed transactions and discarding the incomplete ones. After this process, the file system structure is again consistent. The KFAT[8] file system also applied this method. As metadata in log-based file system is written twice, this technique also suffers from performance problem. Instead of adding a log to the traditional structure of a file system, a log-structured file system, such as Journaling Flash File System 2 (JFFS2)[9] is itself a huge log as a whole. All file updates are just appended to the log, but need not be committed for a second time. However, systems using such log-structured file system suffer from excessively long boot time. To build the index of the file system's current state, the whole volume must be scanned when it is mounted, which is considered intolerable for big volumes.

DSW

The Delay Sequential Write (DSW)[10] method is similar to synchronous write and soft updates. It also keeps the order in which the metadata updates of one file operation are applied to the storage. This means that it has the same level of reliability as a file system using synchronous write or soft updates, i.e. the structure of the file system is kept consistent across crashes, with the exception of possibly a few leaked blocks. The leaked blocks can be reclaimed using a background process. Meanwhile, neither does the DSW method need to write the updated buffers synchronously, nor is it as complicated as soft updates. In fact, the DSW method is both simple and efficient.

Other Approaches

Transaction-safe FAT (TFAT)[11] is a more reliable version of the FAT file system that uses a separate backup FAT table that is not updated until the working copy is back into consistent state after a transaction is done. It also uses FAT chain rerouting to protect file content in addition to metadata. It might be worth mentioning here the Extended File Allocation File System (exFAT)[12] is very promising for portable digital systems, offering better scalability, allocation performance, and reliability and access control than the original FAT file system. The exFAT uses an approach that is similar to TFAT to improve its reliability. M. Baker et al. proposed an alternative approach[13] to implement file system reliability in 1992, by using battery powered non-volatile memory as the buffer cache. Since buffers in the cache are not lost even after a crash, the problem is solved from its root. However, the need of extra hardware restricted this approach's application.

CONCLUSION

In this paper, we discussed the reliability problem of the disk file systems and presented some well established and notable techniques which ensure reliability of disk file systems across system crashes.

REFERENCES

1. Margo I. Seltzer, Gregory R. Ganger, M. Kirk McKusick, Keith A. Smith, Craig A.N. Soules and Christopher A. Stein, "Journaling Versus Soft Updates: Asynchronous Meta-data Protection in File Systems", USENIX Annual Technical Conference (San Diego, CA), 18-23 (2000).
2. W. A. Bhat and S. M. K. Quadri, "Review of FAT Data Structure of FAT32 file system", *Oriental Journal of Computer Science & Technology*, **3**,(1) (2009).
3. Bellevue Linux Users Group, "The Linux Information Project, Ext2fs Definition", http://www.linfo.org/ext2.html
4. M. K. McKusick, "Fsck - The UNIX File System Check Program", In 4.4 BSD System Manager's Manual, O' Reilley & Associates Inc, (1994).
5. M. K. McKusick, W. N. Joy, S. J. Leffler and R. S. Fabry, "A Fast File System for UNIX", In ACM Transactions of computer Systems, **2**(3): (1984).
6. M. K. McKusick and G. R. Ganger, "Soft Updates: A Technique for Eliminating Most Synchronous Write in the Fast File system", In Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference, 1-17 (1999).
7. S. Tweedie, "Journaling the Linux ext2fs Filesystem", In LinuxExpo '98 (1998).
8. 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 2005 US-Korea Conference, ZCTS-142, (2005).
9. D. Woodhouse, "JFFS: The Journaling Flash File System", In Ottawa Linux Symposium. RedHat Inc., (2001).
10. Liang Alei, Liu Kejia, Li Xiaoyong , "FATTY : A reliable FAT File System", Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, Pages: 390-395, Year of Publication: (2007).
11. Microsoft Corp, "Transaction-Safe FAT File System",http://msdn2.microsoft.c0m/en-us / library/aa911939.aspx, (2007).
12. Microsoft Corp, "Extended FAT File System",http://msdn2.rnicrosoft.com/en-us/ library/aa914353.aspx, (2007).
13. M. Baker, S. Asami, E. Deprit, J. Ousterhout and M. Seltzer, "Non-Volatile Memory for Fast, Reliable File Systems", In Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS), **27**(9): (1992).