Study of Techniques for Checking the Consistency in File System

Aniket G. Meshram
Department of Computer Engineering,
Pimpri Chinchwad College of Engineering, Pune.

Sonal Gore
Assisant Profressor,
Department of Computer Engineering,
Pimpri Chinchwad College of Engineering, Pune.

ABSTRACT
File Systems today have grown from a minimal software to a sophisticated system code that is much robust than it was a few years ago. However, still there are issues with the file system design that lead to system crashes and failures. Maintaining file system consistency, even in the face of these crashes remains a subject of study. Like any other information, the metadata information within a file system is a critical aspect that requires attention. In order to maintain file system consistency, it is necessary that the operations the file system carries out be without any bugs. Solutions such as the use of the fsck tool, along with techniques such as journaling and copy-on-write provide solutions only when the user is not using the system. This drawback can be overcome with the concept of runtime checking, but however, deciding the nature of parameter that need to be checked during runtime remains one of the major problems faced by file system experts. In this paper, a discussion is presented of what parameters are required to be checked at runtime describing a way to define those parameters. These parameters can be referred to as declarative consistency rules that can be checked at runtime.

Keywords
File system consistency, Metadata updates, file system checker, journaling, copy-on-write, runtime checking, file system bugs.

1. INTRODUCTION
How does one ensure that the file system is consistent? This is a question that has been bugging the file system experts for a long time. It is clear now that file systems have bugs. Various file system operations have shown the existence of bugs that can corrupt the system and create many problems. A study of file systems bugs has shown that bugs that relate to metadata corruption create serious problems to file system metadata which can be disastrous for organizations that carry out heavy duty operations. Even a small bug into the system can lead to a major crisis for company's working day and night.

UNIX experts had found a solution for this which came out as a tool called the 'fsck' utility tool. This tool is an offline checker that works around the file system to find and repair any bugs that it encounters during its operation. However there are many situations where this offline checker fails to maintain consistency and may produce insecure repairs [14]. It has been shown that fsck may in some cases (instead of repairing) introduce bugs into the system. Further, fsck processing is rather a very slow operation that can take a lot of time for repairing systems leading to a significant downtime. System that operates on large file systems will find using this tool very time consuming and very cumbersome.

However, there are other techniques that are developed to minimize the use of fsck namely journaling and shadow paging. These techniques are well known for creating logs for each transaction done by the file system. The file system checker then makes use of this log that currently holds the information where corruption might be possible and repairs them. Though this works out as a good solution for maintaining consistency they are still error prone since, if bugs propagate to a log they can make the file system prone to failure and crash the system eventually.

Recent study has introduced the technique of a runtime checker that operates even when the user is using the file system. This runtime checking can provide a standard solution to maintaining consistency even when the system is 'live' with the user. However, still a question remains as to how to define rules that can be checked online with the system, what are the consistency rules that can be checked by the runtime checker, and how do we optimize those rules so as to speed up the runtime checking process. Next we describe what sort of consistency checks are performed by the tools and methods discussed before viz. the file system checker tool (fsck), the methods like journaling and the copy-on-write methods. We then go on to describe the related work that has been carried out in this area.

2. LITERATURE SURVEY
2.1. File System Checker (fsck)
Fsck is a very comprehensive tool that checks almost every bit and inch of the file system looking for bugs and repairing them accordingly based on the repair methods defined within it. Fsck is said to normally run in non-interactive mode. But in cases when it seems that a human intervention is required it goes interactive. The primary requirement of the fsck checker is that file system should be unmounted before performing any operations on it. It internally mounts the file system with its mounting handlers and locks the file system for the checking and repairing purpose. The checks that it performs can be summarized as below:

2.1.1. Superblock Checking
This involves checking the superblock for any inconsistencies in file-system size, number of inodes, free-block count, and the free-inode count. In order to do that it scans the entire file system and collect this information in its structure. Each Block Group in the file system structure begins with a superblock. This superblock contains all the metadata information for that block group in the structure. A structure is maintained within the file system that scans all the superblock parameters for consistency along with the parameters mentioned above.

2.1.2. Inode State Checking
Fsck sequentially checks the inode list in the file system from inode 2 (since inode 0 and inode 1 are file system reserved) till the last inode for any discrepancy. In an inode structure there is a mode field as shown in Figure 1. Based on this mode and the allocation information the inode is checked for correctness. If the allocation states that it is neither unallocated nor allocated, then it is considered as bad data and may be cleared.

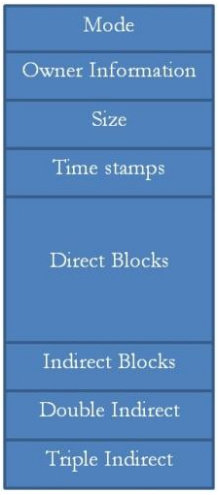

Fig 1. A Typical Inode Structure

2.1.3. Inode Link Checking

Fsck checks the number of links each inode contains starting from the root directory and counting as it continues checking downwards till the current file position the file system hierarchy. If this count does not match the link count specified in the inode structure then the inode is marked as not updated and updates are done based on the new values retrieved from the recent transactions and use operations.

2.1.4. Free Block Checking

This involves checking all the blocks that are marked free in the cylinder group block. If this free block is held by any files then the allocations are rebuilt again. Next, the fsck tool checks if the free block plus the inode block are equal to the total number of blocks in the file system. Further this count is also checked against the summary information within the super-block that counts the total number of free blocks in the file system.

2.1.5. Directory Checks

Fsck checks if the directory marked with '.' has an entry in the directory data block. This entry should be first entry followed by the '..' entry which account for the immediate parent entry for that file. Further, fsck checks that if directories are not linked anywhere into the file system, it links the directory back in a special directory called the *lost & found* directory.

2.2. Journaling

Journaling is nothing but maintaining a log in the file system that contains information about the transactions that take place in the file system. It maintains information about the changes that take place while transactions occur. These changes are typically logged before any transactions are committed into the file system. An ext3 file system is the one that came with the journaling capabilities. A journal was added to the structure just after the superblock as shown in Figure 2.

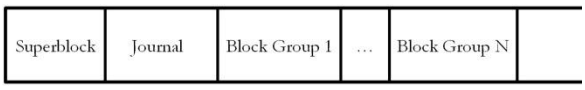

Fig 2. Journal Structure in an ext3 file system

A journal maintain two blocks that keeps track of changes viz. the transaction begin block and the final block that mark the end of transaction. These blocks along with the metadata updates are written to disk and checkpointed. Checkpointing includes writing metadata as well as all the other datablock updates into the right places in the file system.

Journal works as a helping hand for the file system checker, which would rather have scanned the entire file system for bugs, wasting lots of business hours. For example, suppose that the system crash occur after the transaction is written to the logs. In this case suppose that checking is also not done then recovery is performed as follows. When the system restarts, the file system checker (fsck) checks for any discrepancies that might have occurred the last time file system was active. In this case, since the system was crashed earlier a log was maintained. Using this log, a redo logging operation is performed to repeat the same operation for update data and metadata on the final on-disk locations.

2.3. Copy-on-Write

This technique is similar to the journaling technique, except that it uses the concept of one-time sequential write to update on-disk data structures. In this method, when writing to disk, first all the updates including the metadata updates are buffered in the memory segment. When the segment becomes full all this segment information is written to the disk in one single stroke to a free segment without overwriting any existing data structures on the disk.

3. RELATED WORK

3.1. SQCK Checker

Gunawi, et al., has proposed a file system checker based on a declarative query language called SQCK. Borrowing heavily from the database community, SQCK employs declarative queries to check and repair a file system image. The purpose of the e2fsck utility is to check and repair the data structures of an ext2/ext3 file system on disk; in the ideal case, the repaired file system is readable, writable, and contains all of the directories, files, and data of the original file system. E2fsck is a tool that contains more than 30,000 lines of C code and can identify and return 269 different error codes. SQCK is built around five components namely; the *scanner* that reads the relevant portions of the file system from the disk, the *loader* that loads the corresponding information into the database *tables*. The *checker* that is then responsible for running the declarative queries that both check and repair the file system structures and the *flusher* which completes the loop by writing out the changes to disk.

3.2. Runtime Checking

An excellent technique has been presented by Daniel fryer et al. [3] for performing consistency checking operations on a live user system. The techniques spans across procedures from when to check to how to check with the help of what is called a change record. They argue that transaction commit points are well-defined point at which a file system claims to be consistent. The idea of runtime checking works around checking consistency rules at this transaction commit point before the transactions commits itself. Below we describe the details of how, when and what is checked during runtime.

3.2.1. Consistency Properties to Check at Runtime

The consistency properties to check at runtime are derived from the fsck checker itself. But however they are modified so as to create an invariant rule that can save runtime operation. The invariants are declare so that they can be checked at runtime rather than doing a full disk scan. For example, consider a consistency rule that "all live data blocks are marked in the block bitmap". An invariant for such a rule would be to check only the block pointers and the block bitmaps. As they are the only structure that are updated for that rule, checking them at runtime will be effective.

3.2.2. When to Check for Consistency Properties

When a user is live with the system, it is clear that the data or the metadata for that matter would be in an inconsistent state. The in-memory blocks cannot be checked at such state since that would obviously lead to inconsistency, as one is unaware if these copies would get modified or not. Instead one can check for inconsistencies at a point when the file system themselves settles for a consistent state within transaction operations. For journaling, these are the transaction commit points at which the file system is consistent. At this point, a runtime check can be performed to scan the necessary parameters before they commit, so that the updated blocks on the on-disk data structures remain consistent. For example, in shadow paging systems, superblocks are updated after all the transactions are committed to disk.

3.2.3. Data Structure used for Checking Consistency

As mentioned earlier, a change record is maintained, that updates the data structures on the on-disk storage. The format of the change records can be shown as below:

[type, id, field, oldval, newval]

where,

type field mentions the data structure (e.g., inode, directory block).

id is the unique identifier of a specific object of the given type (e.g. inode number). The (type, id) pair locates the specific data structure in the file system image.

field is a field in the structure (e.g. inode size field) or a key from a set (e.g. directory entry name).

oldval and *newval* are the old and new values of the corresponding field.

4. CONCLUSION AND FUTURE SCOPE

Maintaining file system consistency is a very complex issue and requires an extensive study in understanding the file system structure. Moreover, a deep study is required for understanding the correct behaviour for file system. Even in cases where the existing tools and method lack techniques in maintaining file system consistency, runtime checking can play a major role in establishing an effective system. The only question remains is up to what extent this runtime checker maintains consistency and how much can we rely on such techniques for a clean, bug free systems.

5. REFERENCES

[1] Checking the Integrity of Transactional Mechanisms, Daniel Fryer, Mike Qin, Jack Sun, Kah Wai Lee, Angela Demke Brown, Ashvin Goel University of Toronto

[2] A Study of Linux File System Evolution, Lanyue Lu, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Shan Lu Computer Sciences Department, University of Wisconsin, Madison

[3] Recon: Verifying File System Consistency at Runtime, Daniel Fryer, Kuei Sun, Rahat Mahmood, TingHao Cheng, Shaun Benjamin, Ashvin Goel, Angela Demke Brown University of Toronto

[4] Fsck: The UNIX File System Check Program, Marshall Kirk McKusick Computer Science Division Department of Electrical Engineering and Computer Science University of California, Berkeley T. J. Kowalski Bell Laboratories New Jersey

[5] Using Declarative Invariants for Protecting File-System Integrity, Jack Sun, Daniel Fryer, Ashvin Goel and Angela Demke Brown University of Toronto

[6] Journaling the Linux ext2fs Filesystem, Stephen C. Tweedie

[7] Anatomy of Linux journaling file systems, M. Tim Jones, Consultant Engineer Emulex Corp.

[8] Analysis and Evolution of Journaling File Systems, Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau Computer Sciences Department University of Wisconsin, Madison

[9] "Soft Updates: A Solution to the Metadata Update Problem in File Systems", Gregory R. Ganger, Marshall Kirk Mckusick, Craig A. N. Soules, Yale N. Patt

[10] ffsck: The Fast File System Checker, Ao Ma, Chris Dragga, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

[11] SQCK: A Declarative File System Checker, Haryadi S. Gunawi, Abhishek Rajimwale, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau.