UCB CS 162 - Operating Systems and Systems Programming "Lecture 22 - Transactions, End-to-End Arguments, Distributed Decision Making"

Transactions, End-to-End Arguments, Distributed Decision Making

Transactions in a file system are a mechanism to improve the reliability of the system. They extend the concept of atomic updates from memory to persistent storage. A transaction is an atomic sequence of writes that takes the file system from one state to another. If the system crashes before the transaction is complete, none of the writes in the transaction make it through and the file system is restored to the state it was in before the transaction.

Transactions are generally implemented using a log-based approach. There is a log which supports only one atomic operation - appending to it. When we write to a file as part of a transaction, all the individual steps of the write are written as instructions in the log. Once all the instructions are written, we write a "commit" instruction to the log, which says that the transaction is complete and the changes can be applied to the file system.

The OS then goes through the instructions in the log and applies them one by one. If we crash at any time before reaching the commit instruction, we just start from the first instruction and perform the writes again. This is okay because we're just overwriting the same data. If we crash before the commit message is written to the log, all the instructions for this transaction are just thrown out from the log and we say we rolled back.

We use NVRAM for the log to make sure unapplied commits are not lost in case of power failures.

A file system that uses a log in this way to support transactions is called a transactional file system. A transactional file system can be log structured or journaled. In a log structured file system, the data stays in log form. In a journaled file system, the log is only used for recovery and all data is written directly to the disk.

Log Structured File System (LFS)

If we take the concept of a log further, we can make the log itself the file system. It is one continuous sequence of blocks that data is just appended, not modified. As actions are taken (files or directories are created, read, deleted), those actions are also appended to the log. The present state of the file system is then inferred by replaying the actions stored in the log.

As you might expect, this type of filesystem is efficient for writes but not so efficient for reads. However, a good block cache with enough space can make this file system feasible, especially when used with flash memory instead of a hard disk drive. An implementation of a log structured file system is F2FS, a flash file system used in many mobile devices, including the Pixel 3.