File System Crash Recovery

Mendel Rosenblum

File System Crash Recovery

Optional readings: Operating Systems: Principles and Practice:

- Chapter 14 up through Section 14.1

Operating System Crash Recovery

- Crash recovery isn't hard for most of an operating system
  - Everything is wiped out on reboot
  - Just start again with a clean slate
- Doesn't work for a file system:
  - People expect information on disks to survive crashes

Today's topic: how to recover from crashes without losing file data

Challenges for file system crash recovery

- **Data lost**
  - Recent changes may not have been written to disk yet
  - Delay writes: Original Unix: up to 30 seconds worth of changes
- **Inconsistency**
  - Crash in the middle of a modification that affects multiple blocks
  - Examples:
    - Adding block to file: free map was updated, but inode updated to point to block
    - Creating link to a file: new directory entry created, but reference count not updated
  - Ideally disk would give us atomic multiblock operations, but they don't
- The block cache may reorder writes

Approach #1: repair after crash

- Example: Unix `fsck` ("file system check")
  - During every system system startup `fsck` is executed
  - Checks to see if system was shut down cleanly; if so, no more work to do
  - Otherwise, scan file system: identify and fix inconsistencies
- How to tell if the disk shut down cleanly?
  - Have shutdown code write a clean bit to file system when done
- How to identify and fix problems?
  - Read all of file system metadata looking for inconsistencies and repair

`fsck` consistency checks and repair

- Read metadata
  - Inodes, Indirect, and doubly indirect blocks
  - Free map
  - Directories
- Look for inconsistencies and repair

fsck: examples of inconsistencies

- Disk block in an inode but also the free map
  - How could this happen?
  - What happens if we don't fix?
  - How to fix?
  - Fix: remove from free map
- Inode reference count doesn't match links in directories
  - How could this happen? What happens if we don't fix? How to fix?
  - Fix: correct reference count

`fsck`: a harder inconsistency

- Disk block part of two different inodes
  - How could this happen?
    - Delete file A (inode, data block, directory entry, free map)
    - Create file B (inode, data block, directory entry, free map)
    - Delay writeback happens
  - What if we don't fix?
- Fix options:
  - Randomly pick one of the files to own the block, remove it from the other?
  - Make a copy of the block so both files can keep it?
  - Remove from both?

`fsck`: a harder inconsistency 2

- Inode has a reference count > 0 but is not referenced in any directory
  - How could this happen?
  - What if we don't fix?
  - Have file contents but not its name
  - Fix: Create link in a special `/lost+found` directory

Limitations of `fsck`

- Restores disk to consistency, but doesn't prevent loss of information
  - System could end up unusable after recovery
  - Consider a high-level directory gets corrupted
- Security issues: a block could migrate from the password file
- Can take a long time: can't restart system until fsck completes
  - As disks get larger, recovery time increases
  - Became intolerably
- Wouldn't work on modern disk: 5TB
  - Read entire disk: 8 hours
  - Read 10% randomly: few weeks

Approach #2: ordered writes

- Prevent some inconsistencies by making updates in a particular order
  - Choose less harmful inconsistencies
- Example:
  - Adding a block to a file, do disk writes in the following order:
    i. Write back the free map block
    ii. Write the inode, referring to the new block.
- What can we say about the system state after a crash?
  - Can't ever have a block in both the free map and a inode
  - Could leak disk blocks if crash occurs after the free map before inode write
  - Leaks are less serious problem than inconsistencies

General approach to ordered writes

- Make sure info is properly initialized before storing a pointer to it
  - Example: indirect blocks initialized before inode updated to point at it
- Nullify existing pointers to a resource before reusing the resource
  - Example: recall the inconsistency of the same block in two files

Result: no need to wait for `fsck` when rebooting

Problems of ordered writes

- Can leak resources
  - Can run `fsck` in background to recover lost resources
- Easy way of ordering writes is synchronous writes (write-through cache)
  - Slows down file operations

Improved ordered writes design

- Record ordering dependencies in the buffer cache
  - Don't need synchronous writes anymore
  - For example: adding a block to a file
    - Add dependency between inode block and free map block
    - When it's time to write the inode back to disk, push free map block first.
- Tricky to get right: circular dependencies between blocks
  - Need to force writes to prevent cycles

Approach #3: write-ahead logging

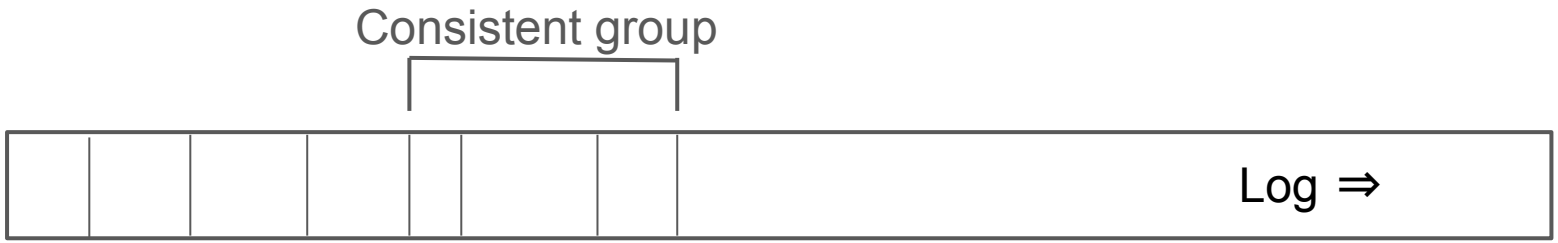
- Also called **journaling file systems**
  - Implemented in Linux ext4 and NTFS (Windows).
- Inconsistencies corrected quickly during reboots
  - Approach long used in database systems
- Approach: Before performing an operation:
  - Record information about the operation in a special append-only log file
  - Actual block updates can be carried out later, in any order
  - Crash recovery: replay the log completing all updates
- Guarantees that once an operation is started, it will eventually complete

How do log entries describe changes?

- One approach: log entries describe logical operations
  - Add block 99421 to inode 862 as block index 93
- Another approach: log physical disk updates
  - Patch 4 bytes at offset 324 in block 6159972 with the value 9942
- Log entries must be **idempotent** (could get replayed multiple times)
  - Handling of crashes during crash recovery tricky otherwise

Log consistent groups

- Can have multiple log entries as part of the same logical operation
  - Ultimately want crash recovery to apply all or none of the entries
- Need a way of identify consistent groups in log
  - For example: All the changes required to add a block to a file
  - Don't process any of the entries in a group unless all are present



Assignment 8: Journaling File system

- Uses a hybrid log entry formats:
  - Patch disk blocks
  - Mark blocks free/allocated
- Has a consistent group mechanism: transactions
  - Mark log entries with start and end transaction

Problems with write-ahead logging

- Log grows over time, so recovery could be slow
  - Need to be able to truncate log
- Solution: occasional **checkpoints**
  - Record current log head (just past last complete transaction)
  - Flush all dirty blocks to disk
- After a checkpoint
  - Truncate log to the recorded position
- How much to log?
  - Typically the log is used only for metadata (free map, inodes, indirect blocks)
  - Logging all file data is much more expensive

Logging Advantages and Disadvantages

- Advantages
  - Fast recovery
  - Eliminates inconsistencies
  - Log written sequentially, fast no seeks
  - Safe to use delay writes on metadata
- Disadvantages
  - Synchronous disk write before every metadata operation
  - Can still lose data after crashes (due to delayed writes)

Delayed log writes

- Safety requirement:
  - Log entry must be written before any other disk blocks related to the log entry
  - Not required to be synchronous log writes
- Delay log writes
  - Buffer log in memory
  - Before writing back a cache block, flush log
- Separates durability from consistency

Remaining problems with write-ahead logging

- Desirable to use delayed writes for performance
  - Crashes can loose recently-written data if it hasn't between flushed to disk
  - Solution: apps can use `fsync` to force data to disk
- Disks fail
  - One of the greatest causes of problems in large datacenters
  - Solution: replication or backup copies

Conclusions

- Care about:
  - Performance
  - Durability
  - Consistency
- Interesting tradeoffs: can't get all three
  - To get highest performance, must give up some crash recovery capability.
- Must decide what kinds of failures you want to recover from.