


COS 318: Operating Systems

File Systems Reliability and Performance (Contd.)


Jaswinder Pal Singh
 Computer Science Department
 Princeton University

<http://www.cs.princeton.edu/courses/cos318/>



Topics

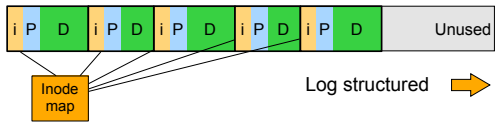
- Journaling and LFS
- Copy on Write and Write Anywhere (NetApp WAFL)




2

Log-structured File System (LFS)

- Structure the entire file system as a log with segments
 - A segment has i-nodes, indirect blocks, and data blocks
 - An i-node map to map i-node number to i-node locations
 - All writes are sequential
- Issues
 - There will be holes when deleting files
 - Need garbage collection to get rid of holes
 - Read performance?
- Goal is to improve write performance
 - Not to confuse with the log for transactions/journaling
 - Also useful for write and wear-leveling with NAND Flash




Log structured →



Revisit Implementation of Transactions

- BeginTransaction
 - Start using a "write-ahead" log on disk
 - Log all updates
- Commit
 - Write "commit" at the end of the log
 - Then "write-behind" to disk by writing updates to disk
 - Clear the log
- Rollback
 - Clear the log
- Crash recovery
 - If there is no "commit" in the log, do nothing
 - If there is "commit," replay the log and clear the log
- Issues
 - All updates on the log must be idempotent
 - Each transaction has an Id or TID
 - Must have a way to confirm that a disk write completes



4

Journaling File System

- ♦ Consistent updates using transactions
 - Recovery is simple
- ♦ Store the log on disk storage
 - Overhead is high for journaling all updates
 - SW for commodity hardware journaling only metadata (Microsoft NTFS and various Linux file systems)
- ♦ Store the log on NVRAM
 - Efficient to journal all updates
 - Can achieve fast writes (many IOPS)
- ♦ "Write behind" performs real updates
 - Where to update (i-nodes and data blocks)?
 - File layout is critical to performance



5

Journaling File System

- ♦ Example: Append a data block to a file on disk
 - Allocate disk blocks for data and i-node (update bitmap)
 - Update i-node and data blocks
- ♦ Journaling all updates
 - Execute the following transaction:
 - BeginTransaction
 - Update i-node
 - Update bitmap
 - Write data block
 - Commit
- ♦ Journaling only metadata
 - Write data block
 - Execute the following transaction:
 - BeginTransaction
 - Update i-node
 - Update bitmap
 - Commit



6

WAFL (Write Anywhere File Layout)

- ♦ WAFL: Write Anywhere File Layout
 - The basic NetApp file system
- ♦ Design goals
 - Fast services (more operations/sec and higher bandwidth)
 - Support large file systems and allow growing smoothly
 - High-performance software RAID (esp for slow writes due to parity considerations)
 - Restart quickly and consistently after a crash
- ♦ Special features
 - Introduce snapshots, using Copy-on-Write
 - Journaling by using NVRAM to implement write-ahead log
 - Layout inspired by LFS



7

Snapshots

- ♦ A snapshot is a read-only copy of the file system
 - Introduced in 1993
 - It has become a **standard feature** of today's file servers
- ♦ Use snapshots
 - System administrator configures the number and frequency of snapshots
 - An initial system can keep up to 20 snapshots
 - Use snapshots to recover individual files
- ♦ An example


```
phoenix% cd .snapshot
phoenix% ls
hourly.0 hourly.2 hourly.4 nightly.0 nightly.2 weekly.1
hourly.1 hourly.3 hourly.5 nightly.1 weekly.0
phoenix%
```
- ♦ How much space does a snapshot consume?

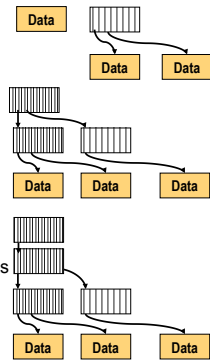


8

i-node, Indirect and Data Blocks

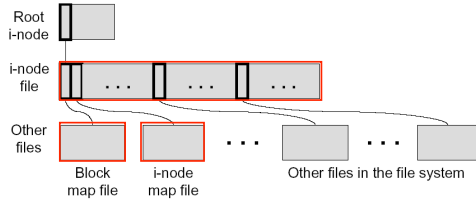
- WAFL uses 4KB blocks
 - i-nodes (evolved from UNIX's)
 - Data blocks
- File size < 64 bytes
 - i-node stores data directly
- File size < 64K bytes
 - i-node stores 16 ptrs to data
- File size < 64M bytes
 - i-node: 16 ptrs to indirect blocks
 - Each stores 1K pointers to data
- File size > 64M bytes
 - i-node: ptrs to doubly indirect blocks

Note: each type points to all blocks at same level



WAFL Layout

- A WAFL file system has
 - A root i-node: root of everything
 - An i-node file: contains all i-nodes
 - A block map file: indicates free blocks
 - An i-node map file: indicates free i-nodes
- } Metadata in files



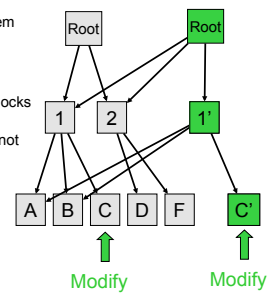
Why Keep Metadata in Files

- Allow meta-data blocks to be written anywhere on disk
 - This is the origin of "Write Anywhere File Layout"
 - Any performance advantage?
- Easy to increase the size of the file system dynamically
 - Add a disk can lead to adding i-nodes
 - Integrate volume manager with WAFL
- Enable copy-on-write to create snapshots
 - Copy-on-write new data and metadata on new disk locations
 - Fixed metadata locations very cumbersome for this
- Any exception to "write anywhere"?



Snapshot Implementation

- WAFL file system is a tree of blocks
- Snapshot step 1
 - Replicate the root i-node
 - New root i-node is the active file system
 - Old root i-node is the snapshot
- Snapshot step 2...n
 - Copy-on-write blocks to the root
 - Active root i-node points to the new blocks
 - Writes to the new block
 - Future writes into the new blocks will not trigger copy-on-write



File System Consistency

- ♦ Create a "consistency point" or hidden snapshot
 - Create a consistency point or snapshot every 10 seconds
 - On a crash, revert the file system to this snapshot
 - Not visible by users
- ♦ Many requests between consistency points
 - Consistency point i
 - Many writes
 - Consistency point i+1 (advanced atomically)
 - Many writes
 - ...



13

Non-Volatile RAM

- ♦ Non-Volatile RAM
 - Flash memory (slower)
 - Battery-backed DRAM (fast but battery lasts for only days)
- ♦ Use an NVRAM to log writes
 - Log all write requests since the last consistency point
 - A clean shutdown empties NVRAM, creates one more snapshot, and turns off NVRAM
 - A crash recovery needs to replay log to recover data from NVRAM to the most recent snapshot and turn on the system
- ♦ Use two logs
 - Buffer one while writing another



14

Write Allocation

- ♦ WAFL can write to any blocks on disk
 - File metadata (i-node file, block map file and i-node map file) are in files
- ♦ WAFL can write blocks in any order
 - Rely on consistency points to enforce file consistency
 - NVRAM to buffer writes to implement ordering
- ♦ WAFL can allocate disk space for many NFS operations at once in a single write episode
 - Reduce the number of disk I/Os
 - Allocate space that is low latency

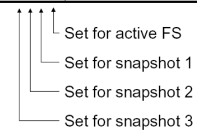


15

Snapshot Data Structure

- ♦ WAFL uses 32-bit entries in block map file
 - 32-bit for each 4K block
 - 32-bit entry = 0: the disk block is free
- ♦ Bit 0 = 1: active file system references the block
- ♦ Bit 1 = 1: the most recent snapshot references the block

Time	Block map entry	Description
T1	0 0 0 0 0 0 0 0	Block is free
T2	0 0 0 0 0 0 0 1	Active FS uses it
T3	0 0 0 0 0 0 1 1	Create snapshot 1
T4	0 0 0 0 0 1 1 1	Create snapshot 2
T5	0 0 0 0 0 1 1 0	Active FS deletes it
T6	0 0 0 0 0 1 0 0	Delete snapshot 1
T7	0 0 0 0 0 0 0 0	Delete snapshot 2



16

Snapshot Creation

- ♦ Problem
 - Many NFS requests may arrive while creating a snapshot
 - File cache may need replacements
 - Undesirable to suspend the NFS request stream
- ♦ WAFL solution
 - Before a creation, mark dirty cache data "in-snapshot" and suspend NFS request stream
 - Defer all modifications to "in-snapshot" data
 - Modify cache data not marked "in-snapshot"
 - Do not flush cache data not marked "in-snapshot"



17

Algorithm

- ♦ Steps
 - Allocate disk space for "in-snapshot" cached i-nodes
 - Copy these i-nodes to disk buffer
 - Clear "in-snapshot" bit of all cached i-nodes
 - Update the block-map file
 - For each entry, copy the bit for active FS to the new snapshot
 - Flush
 - Write all "in-snapshot" disk buffers to their new disk locations
 - Restart NFS request stream
 - Duplicate the root i-node
- ♦ Performance
 - Typically it takes less than a second



18

Snapshot Deletion

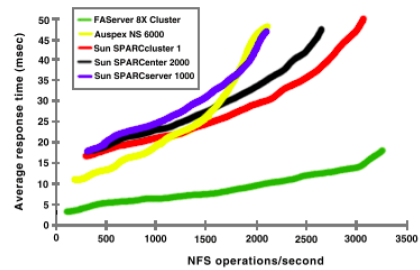
- ♦ Delete a snapshot's root i-node
- ♦ Clear bits in block-map file
 - For each entry in block-map file, clear the bit representing the snapshot



19

Performance

- ♦ SPEC SFS benchmark shows 8X faster than others



20

Summary

- ◆ Journaling and LFS
 - Journaling uses transactions to achieve consistency
 - LFS improves write performance
- ◆ WAFL
 - Write anywhere layout (inspired by LFS)
 - Snapshots have become a standard feature
 - Journaling with NVRAM

