COS 318: Operating Systems

Snapshot and NFS

Andy Bavier
Computer Science Department
Princeton University

http://www.cs.princeton.edu/courses/archive/fall10/cos318/

Topics

- Revisit Transactions and Logging
- NetApp File System
- NFS

Transactions

- Bundle many operations into a transaction
  - One of the first transaction systems is Sabre American Airline reservation system, made by IBM
- Primitives
  - BeginTransaction
    - Mark the beginning of the transaction
  - Commit (End transaction)
    - When transaction is done
  - Rollback (Abort transaction)
    - Undo all the actions since "Begin transaction."
- Rules
  - Transactions can run concurrently
  - Rollback can execute anytime
  - Sophisticated transaction systems allow nested transactions

Implementation

- 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

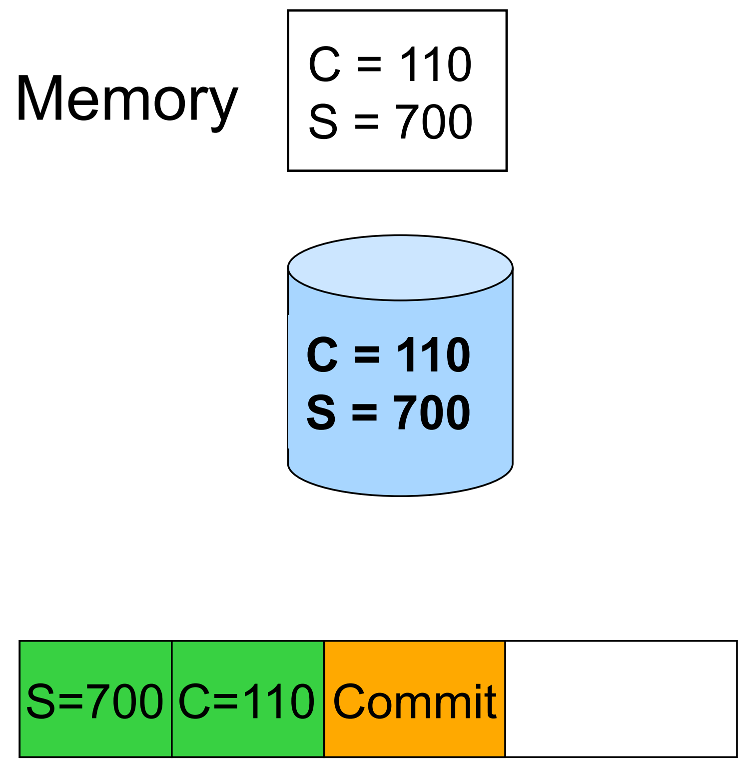
- Assumptions
  - Writing to disk is correct (recall the error detection and correction)
  - Disk is in a good state before we start

An Example: Atomic Money Transfer

- Move $100 from account S to C (1 thread):

```
BeginTransaction
    S = S - $100;
    C = C + $100;
Commit
```

- Steps:
  - 1: Write new value of S to log
  - 2: Write new value of C to log
  - 3: Write commit
  - 4: Write S to disk
  - 5: Write C to disk
  - 6: Clear the log
- Possible crashes
  - After 1
  - After 2
  - After 3 before 4 and 5
- Questions
  - Can we swap 3 with 4?
  - Can we swap 4 and 5?



Revisit The Implementation

- 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

- Questions
  - What if there is a crash during the recovery?

Use Transactions in File Systems

- Make a file operation a transaction
  - Create a file
  - Move a file
  - Write a chunk of data
  - …
  - Would this eliminate any need to run fsck after a crash?
- Make arbitrary number of file operations a transaction
  - Just keep logging but make sure that things are idempotent: making a very long transaction
  - Recovery by replaying the log and correct the file system
  - This is called journaling file system
  - Almost all new file systems are journaling (Windows NTFS, Veritas file system, file systems for Linux)

Issue with Logging: Performance

- For every disk write, we now have two disk writes (on different parts of the disk)?
  - It is not so bad because logging is sequential and write-behind can be done asynchronously.
- Performance tricks
  - Changes made in memory and then logged to disk
  - Logging are sequentially done a different disk.
  - Merge multiple writes to the log with one write
  - Use NVRAM (Non-Volatile RAM) to keep the log

Log Management

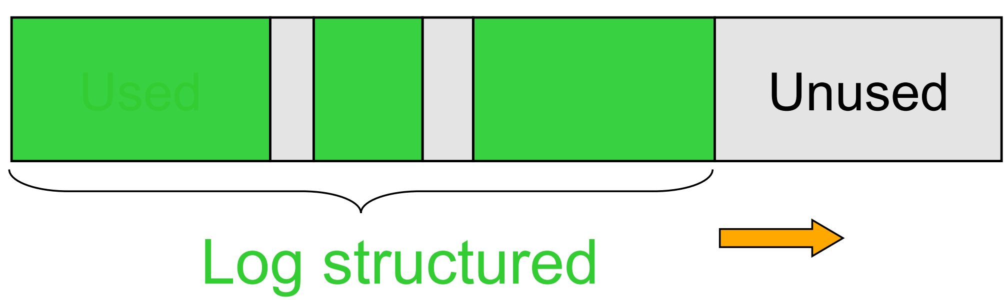
- How big is the log? Same size as the file system?
- Observation
  - Log what's needed for crash recovery
- Management method
  - Checkpoint operation: flush the buffer cache to the log and then write behind
  - After a checkpoint, we can truncate log and start again
  - Log needs to be big enough to hold changes in memory
- Some file systems log only metadata (file descriptors and directories)
  - Would this be a problem?

What to Log?

- Physical blocks (directory blocks and inode blocks)
  - Easy to implement but takes more space
  - Which block image?
    - Before operation: Easy to go backward during recovery
    - After operation: Easy to go forward during recovery.
    - Both: Can go either way.
- Logical operations
  - Example: Add name “foo” to directory #41
  - More compact
  - But more work at recovery time

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
- All writes are sequential (no seeks)
- There will be holes when deleting files
- Questions
  - What about read performance?
  - How would you clean (garbage collection)?



Case: NetApp's NFS File Server

- WAFL: Write Anywhere File Layout
  - The basic NetApp's file system
- Design goals
  - Fast services (fast means more operations/sec and higher bandwidth)
  - Support large file systems and allow growing smoothly
  - High-performance software RAID
  - Restart quickly after a crash
- Special features
  - Introduce snapshots
  - Use NVRAM to reduce latency and maintain consistency

Snapshots

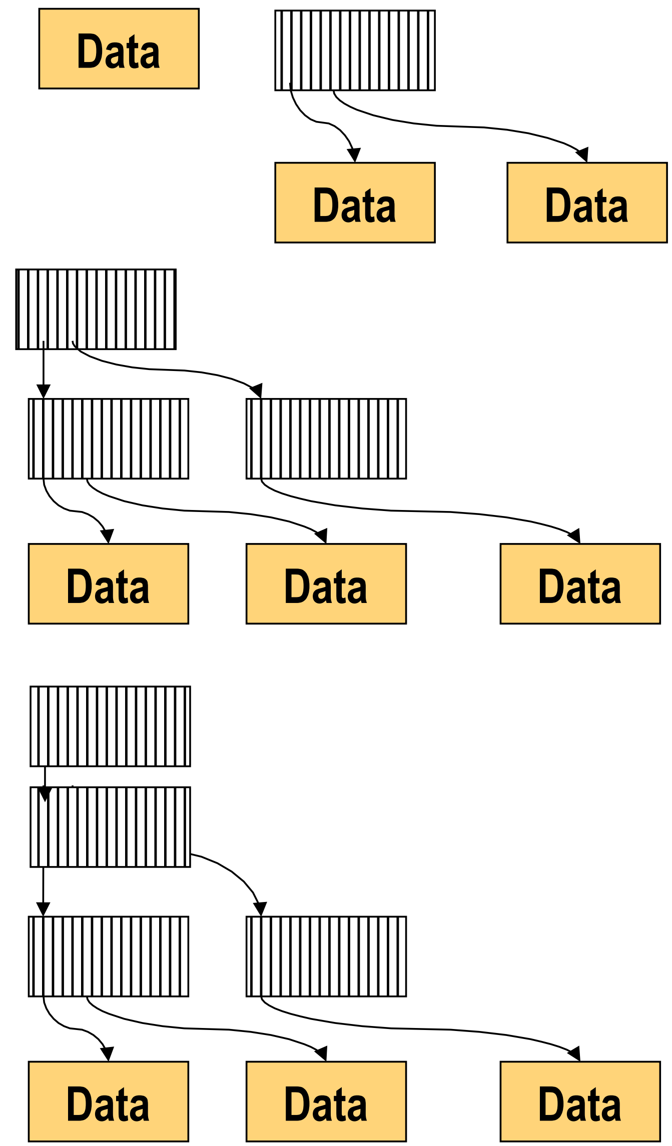
- A snapshot is a read-only copy of the file system
  - Introduced in 1993
  - It has become **a standard feature** of today's file server
- Use snapshots
  - System administrator configures the number and frequency of snapshots
  - An initial system can keep up to 20 snapshots
  - Use snapshots to recover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

```
arizona% cd .snapshot
arizona% ls
hourly.0 hourly.2 hourly.4 nightly.0 nightly.2 weekly.1
hourly.1 hourly.3 hourly.5 nightly.1 weekly.0
arizona%
```

- How much space does a snapshot consume?
  - 10-20% space per week

i-node, Indirect and Data Blocks

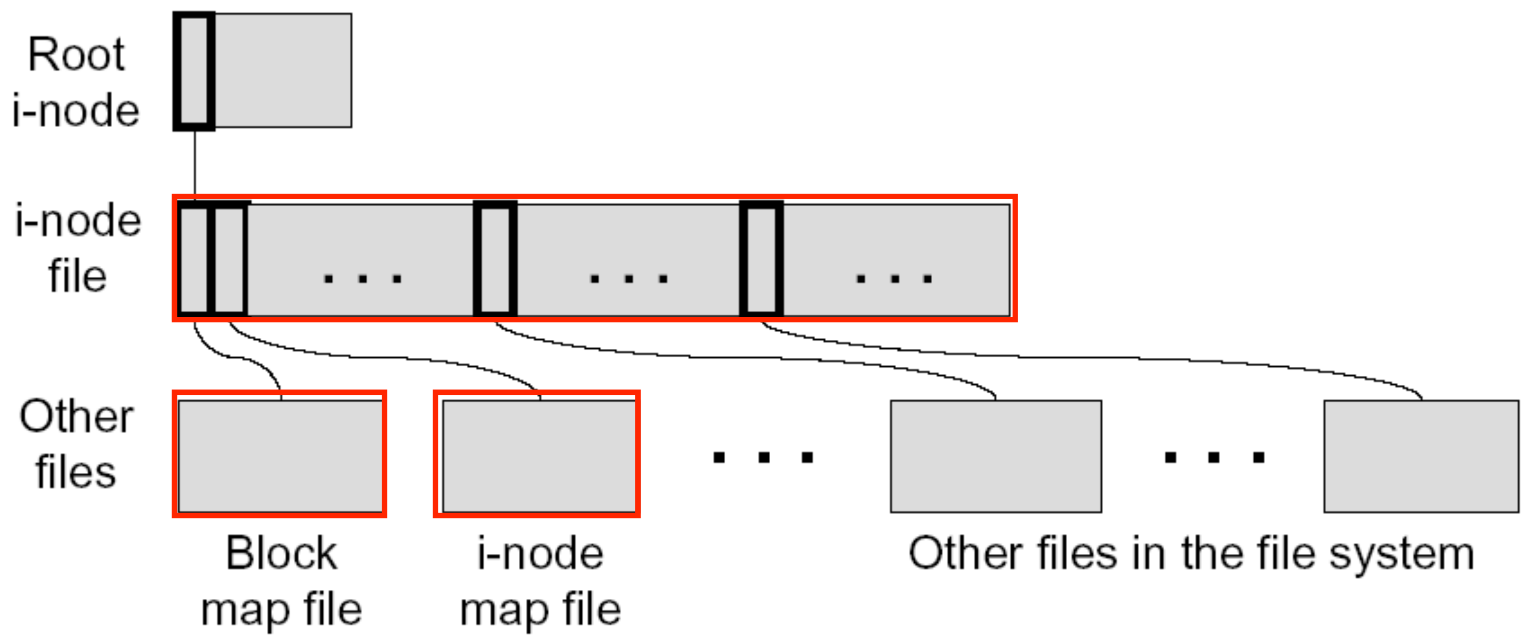
- WAFL uses 4KB blocks
  - i-nodes (evolved from UNIX's)
  - Data blocks
- File size < 64 bytes
  - i-node stores data directly
- File size < 64K bytes
  - i-node stores 16 pointers to data
- File size < 64M bytes
  - i-node stores 16 pointers to indirect blocks
  - Each indirect pointer block stores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



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

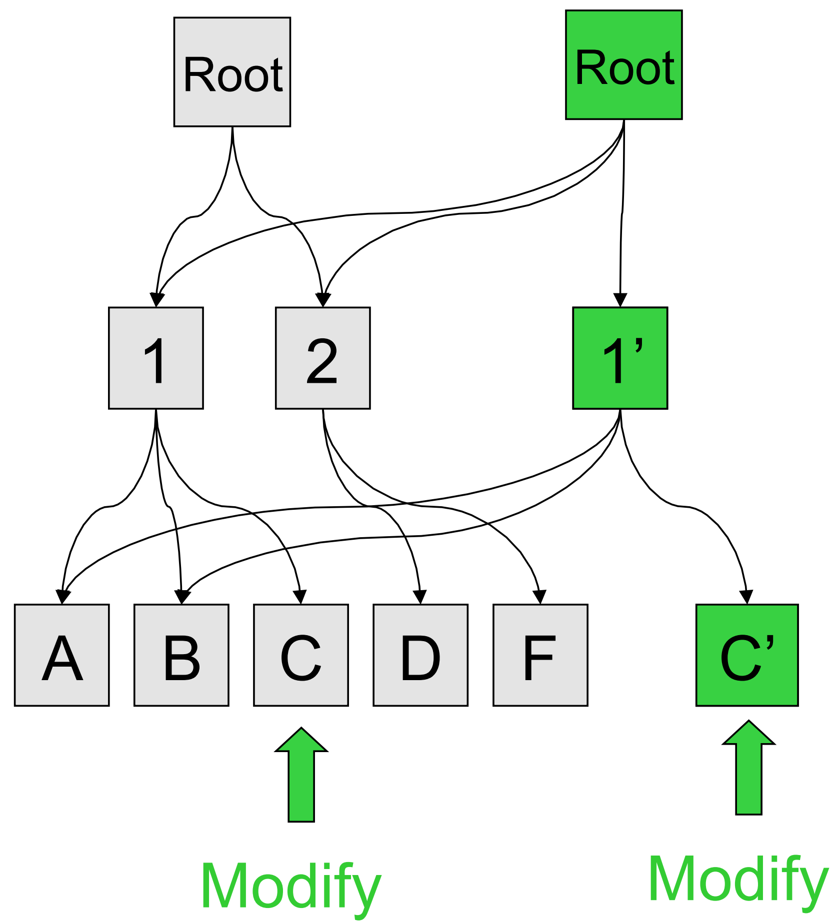


Why Keeping Metadata in Files

- 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 are cumbersome

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
- An "add-on" snapshot mechanism for a traditional file system?



File System Consistency

- Create a 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
  - …
- Question
  - Any relationships with transactions?

Non-Volatile RAM

- Non-Volatile RAM
  - Flash memory (slower)
  - Battery-backed DRAM (fast but battery lasts for only days)
- Use an NVRAM to buffer writes
  - Buffer 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 recover data from NVRAM to the most recent snapshot and turn on the system
- Use two logs
  - Buffer one while writing another
- Issues
  - What is the main disadvantage of NVRAM?
  - How large should the NVRAM be?

Write Allocation

- WAFL can write to any blocks on disk
  - File metadata (i-node file, block map file and i-node map file) is in the file system
- 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
- Issue
  - What about read performance?

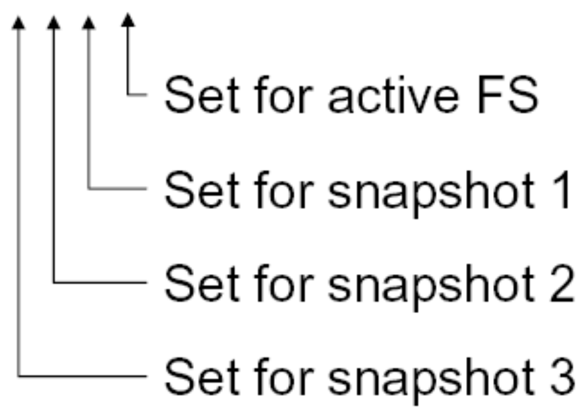
Snapshot Data Structure

- WAFL uses 32-bit entries in the block map file
  - 32-bit for each 4KB disk block
  - 32-bit entry = 0: the 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



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"
  - Defer all modifications to “in-snapshot” data
  - Modify cache data not marked “in-snapshot”
  - Do not flush cache data not marked “in-snapshot”

Algorithm

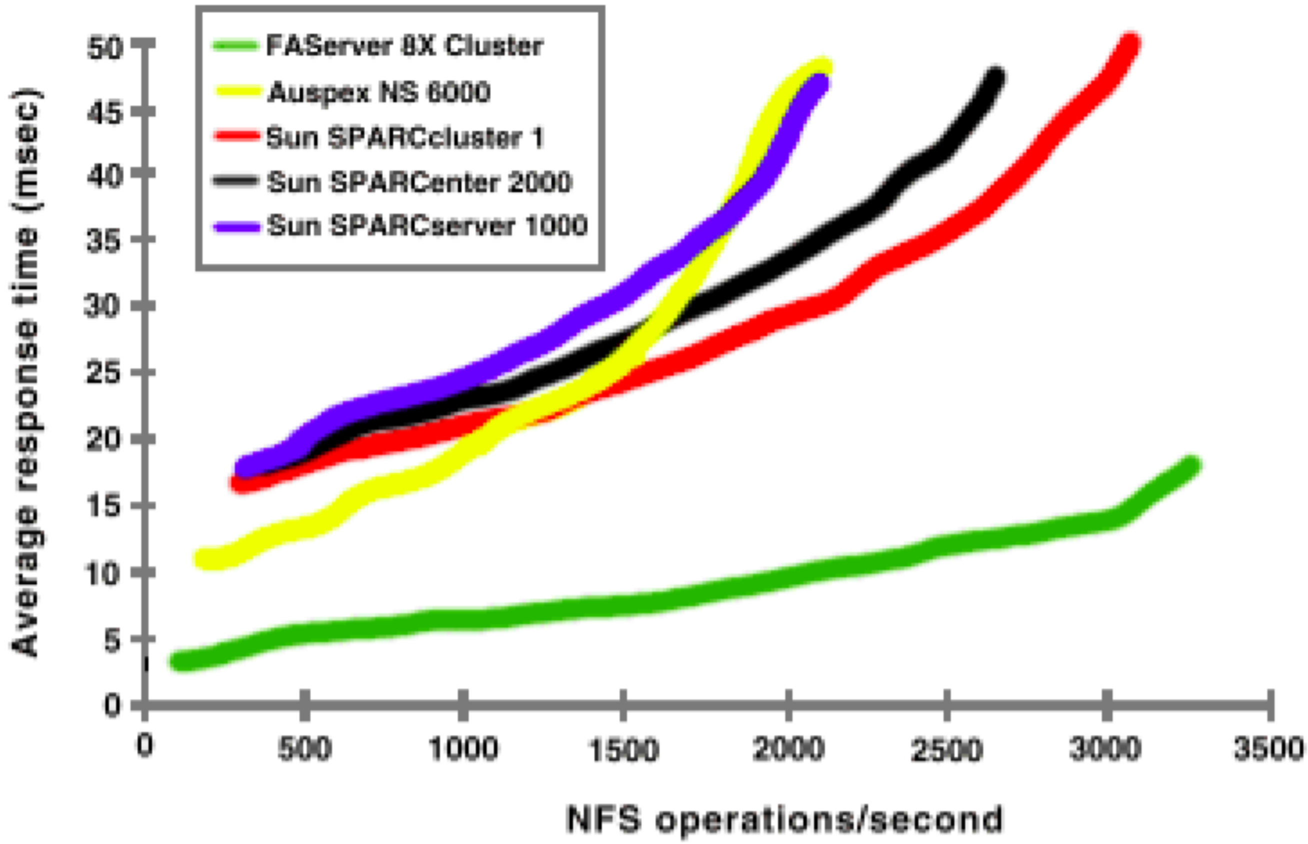
- Steps
  - Allocate disk space for “in-snapshot” blocks
    - Copy “in-snapshot” cached i-nodes to disk buffer
    - Clear “in-snapshot” bit of all cached i-nodes
  - Update the block-map file
    - For each entry, copy the bit for active FS to the new snapshot
  - Flush
    - Write all “in-snapshot”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

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

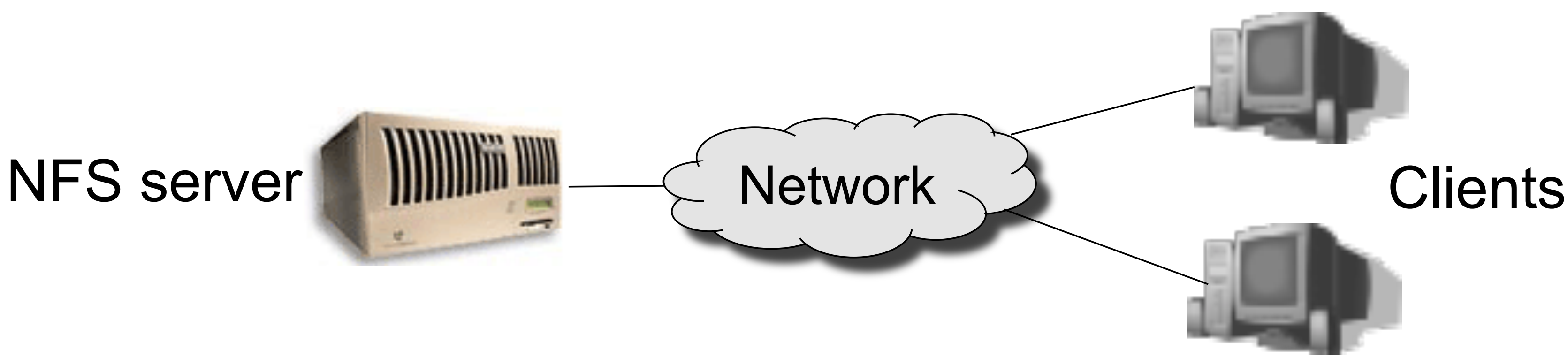
Performance

- SPEC SFS benchmark shows 8X faster than others



Network File System

- Sun introduced NFS v2 in early 80s
- NFS server exports directories to clients
- Clients mount NFS server's exported directories (auto-mount is possible)
- Multiple clients share a NFS server



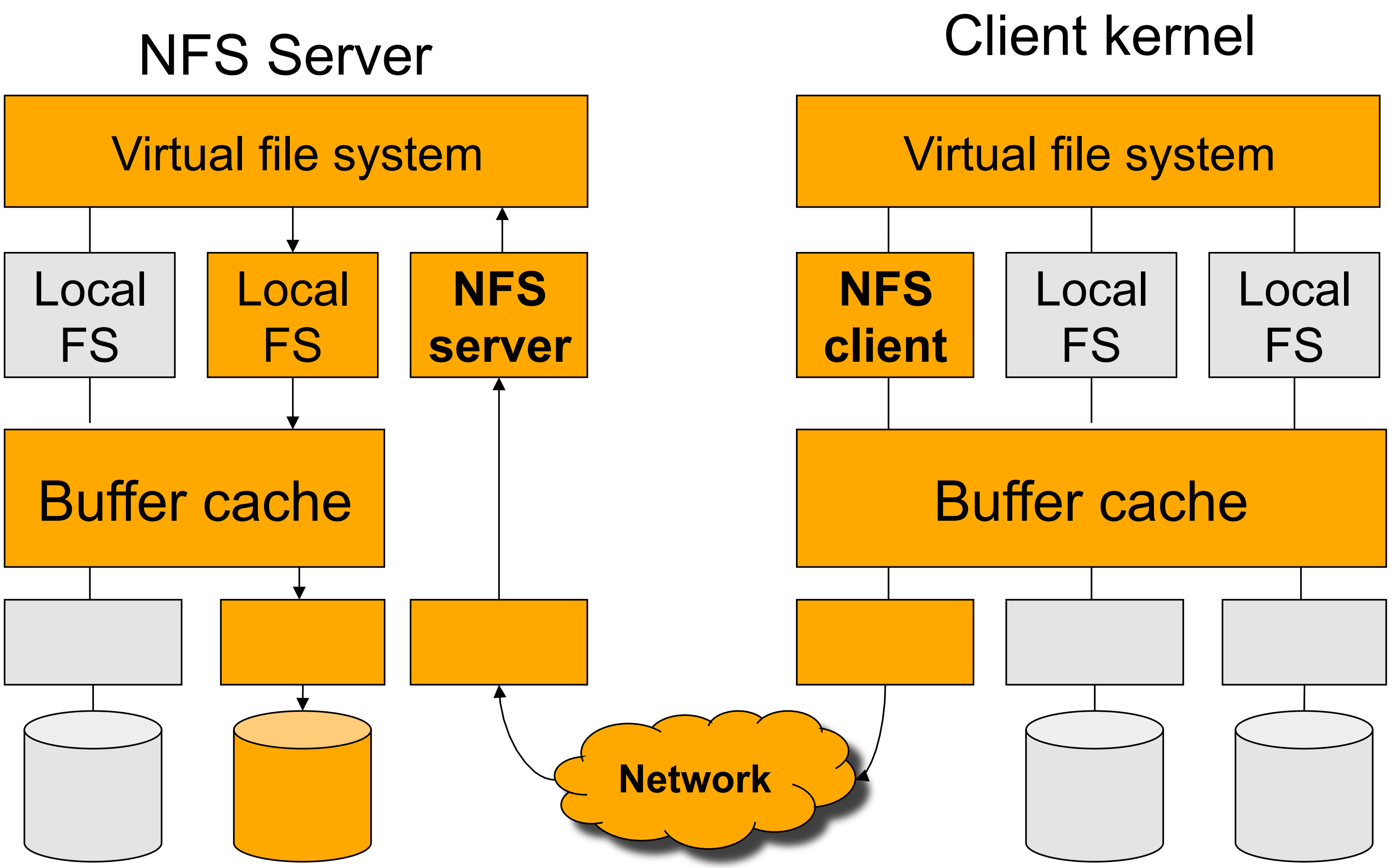
NFS Protocol (v3)

1. NULL: Do nothing
2. GETATTR: Get file attributes
3. SETATTR: Set file attributes
4. LOOKUP: Lookup filename
5. ACCESS: Check Access Permission
6. READLINK: Read from symbolic link
7. READ: Read From file
8. WRITE: Write to file
9. CREATE: Create a file
10. MKDIR: Create a directory
11. SYMLINK: Create a symbolic link
12. MKNOD: Create a special device
13. REMOVE: Remove a File
14. RMDIR: Remove a Directory
15. RENAME: Rename a File or Directory
16. LINK: Create Link to an object
17. READDIR: Read From Directory
18. READDIRPLUS: Extended read from directory
19. FSSTAT: Get dynamic file system information
20. FSINFO: Get static file system Information
21. PATHCONF: Retrieve POSIX information
22. COMMIT: Commit cached data on a server to stable storage

NFS Protocol

- No open and close
- Use a global handle in the protocol
  - Read some bytes
  - Write some bytes
- Questions
  - What is stateless?
  - Is NFS stateless?
  - What is the tradeoffs of stateless vs. stateful?

NFS Implementation



NFS Client Caching Issues

- Client caching
  - Read-only file and directory data (expire in 60 seconds)
  - Data written by the client machine (write back in 30 seconds)
- Consistency issues
  - Multiple client machines can perform writes to their caches
  - Some cache file data only and disable client caching of a file if it is opened by multiple clients
  - Some implement a network lock manager

NFS Protocol Development

- Version 2 issues
  - 18 operations
  - Size: limit to 4GB file size
  - Write performance: server writes data synchronously
  - Several other issues
- Version 3 changes (most products still use this one)
  - 22 operations
  - Size: increase to 64 bit
  - Write performance: WRITE and COMMIT
  - Fixed several other issues
  - Still stateless
- Version 4 changes
  - 42 operations
  - Solve the consistency issues
  - Security issues
  - **Stateful**

Summary

- Consistent updates
  - Transactions use a write-ahead log and write-behind to update
  - Journaling file systems use transactions
- WAFL
  - Write anywhere layout
  - Snapshots have become a standard feature
- NFS
  - Stateless network file system protocol
  - Client and server caching