# Other File Systems: NFS and GFS

Ken Birman

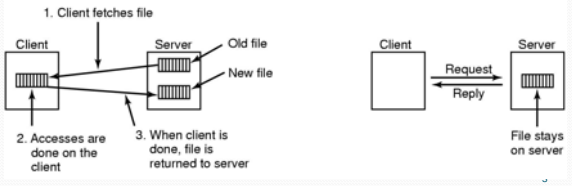
## Distributed File Systems

- Goal: view a distributed system as a file system
  - Storage is distributed
  - Web tries to make world a collection of hyperlinked documents
- Issues not common to usual file systems
  - Naming transparency
  - Load balancing
  - Scalability
  - Location and network transparency
  - Fault tolerance
- We will look at some of these today

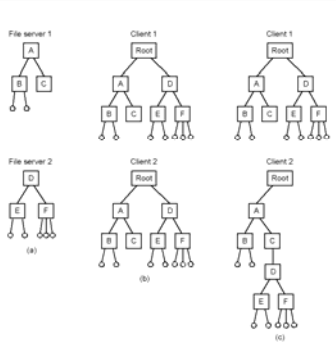


## Transfer Model

- Upload/download Model:
  - Client downloads file, works on it, and writes it back on server
  - Simple and good performance
- Remote Access Model:
  - File only on server; client sends commands to get work done




## Directory Hierarchy





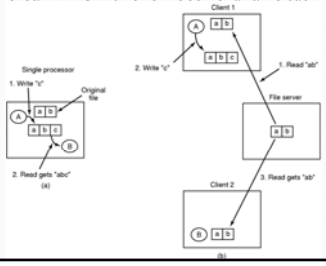

## Naming transparency

- Naming is a mapping from logical to physical objects
- Ideally client interface should be transparent
  - Not distinguish between remote and local files
  - */machine/path* or *mounting remote FS in local hierarchy* are not transparent
- A transparent DFS hides the location of files in system
- 2 forms of transparency:
  - Location transparency: path gives no hint of file location
    - */server1/dir1/dir2/x* tells *x* is on *server1*, but not where *server1* is
  - Location independence: move files without changing names
    - Separate naming hierarchy from storage devices hierarchy



## File Sharing Semantics

- Sequential consistency: reads see previous writes
  - Ordering on all system calls seen by all processors
  - Maintained in single processor systems
  - Can be achieved in DFS with one file server and no caching



## Caching

- Keep repeatedly accessed blocks in cache
  - Improves performance of further accesses
- How it works:
  - If needed block not in cache, it is fetched and cached
  - Accesses performed on local copy
  - One master file copy on server, other copies distributed in DFS
  - Cache consistency problem: how to keep cached copy consistent with master file copy
- Where to cache?
  - Disk: Pros: more reliable, data present locally on recovery
  - Memory: Pros: diskless workstations, quicker data access,
  - Servers maintain cache in memory

7

## File Sharing Semantics

- Other approaches:
  - Write through caches:
    - immediately propagate changes in cache files to server
    - Reliable but poor performance
  - Delayed write:
    - Writes are not propagated immediately, probably on file close
    - Session semantics: write file back on close
    - Alternative: scan cache periodically and flush modified blocks
    - Better performance but poor reliability
  - File Locking:
    - The upload/download model locks a downloaded file
    - Other processes wait for file lock to be released

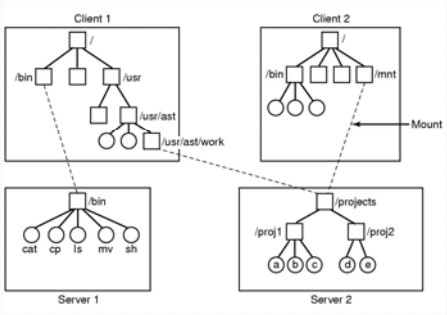
8

## Network File System (NFS)

- Developed by Sun Microsystems in 1984
  - Used to join FSes on multiple computers as one logical whole
- Used commonly today with UNIX systems
- Assumptions
  - Allows arbitrary collection of users to share a file system
  - Clients and servers might be on different LANs
  - Machines can be clients and servers at the same time
- Architecture:
  - A server exports one or more of its directories to remote clients
  - Clients access exported directories by mounting them
    - The contents are then accessed as if they were local

9

## Example




10

## NFS Mount Protocol

- Client sends path name to server with request to mount
  - Not required to specify where to mount
- If path is legal and exported, server returns file handle
  - Contains FS type, disk, i-node number of directory, security info
  - Subsequent accesses from client use file handle
- Mount can be either at boot or automount
  - Using automount, directories are not mounted during boot
  - OS sends a message to servers on first remote file access
  - Automount is helpful since remote dir might not be used at all
- Mount only affects the client view!

11

## NFS Protocol

- Supports directory and file access via RPCs
- All UNIX system calls supported other than *open* & *close*
- *Open* and *close* are intentionally not supported
  - For a *read*, client sends *lookup* message to server
  - Server looks up file and returns handle
  - Unlike *open*, *lookup* does not copy info in internal system tables
  - Subsequently, *read* contains file handle, offset and num bytes
  - Each message is self-contained
- Pros: server is stateless, i.e. no state about open files
- Cons: Locking is difficult, no concurrency control

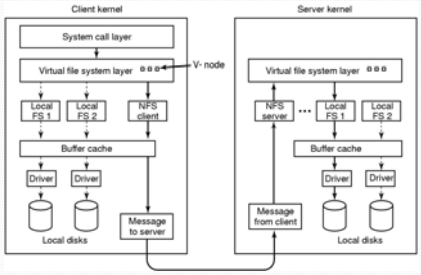
12

## NFS Implementation

- Three main layers:
  - System call layer:
    - Handles calls like open, read and close
  - Virtual File System Layer:
    - Maintains table with one entry (v-node) for each open file
    - v-nodes indicate if file is local or remote
      - If remote it has enough info to access them
      - For local files, FS and i-node are recorded
  - NFS Service Layer:
    - This lowest layer implements the NFS protocol

13

## NFS Layer Structure




14

## How NFS works?

- Mount:
  - Sys ad calls mount program with remote dir, local dir
  - Mount program parses for name of NFS server
  - Contacts server asking for file handle for remote dir
  - If directory exists for remote mounting, server returns handle
  - Client kernel constructs v-node for remote dir
  - Asks NFS client code to construct r-node for file handle
- Open:
  - Kernel realizes that file is on remotely mounted directory
  - Finds r-node in v-node for the directory
  - NFS client code then opens file, enters r-node for file in VFS, and returns file descriptor for remote node

15

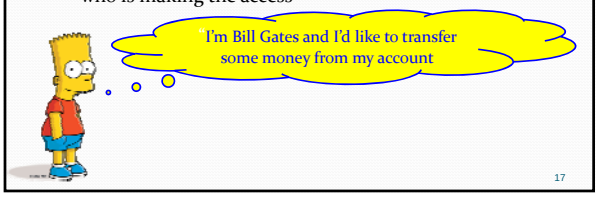
## Cache coherency

- Clients cache file attributes and data
  - If two clients cache the same data, cache coherency is lost
- Solutions:
  - Each cache block has a timer (3 sec for data, 30 sec for dir)
    - Entry is discarded when timer expires
  - On open of cached file, its last modify time on server is checked
    - If cached copy is old, it is discarded
  - Every 30 sec, cache time expires
    - All dirty blocks are written back to the server

16

## Security

- A serious weakness in NFS
  - It has two modes: a secure mode (but turned off by default) and an insecure mode
  - In insecure mode, it trusts the client to tell the server who is making the access




17

## The Google File System

Google

From a slide set by Tim Chuang, U.W. Seattle

## Introduction

First version was developed by Larry Page and Sergey Brin as a virtual file system – BigFiles.

Google File System was developed by Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung to be the successor to BigFiles.

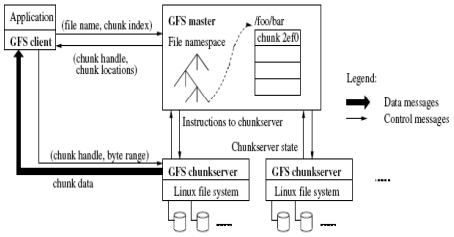
## Design Assumptions

- Built from many inexpensive commodity hardware that "often" fail.
- Store a modest number of large files.
- Support two kinds of reads – large streaming and small random reads.
- Support large, sequential writes that append data to files.
- Support for multiple clients that concurrently append to the same file.

## Architecture

- Single Master, multiple chunkservers.
- Fixed chunk size of 64 MB.
- Periodic communication between master and chunkservers with "HeartBeat" messages.
- No file caching.

## Simple Read Interactions




## Effect?

- User file request is "mapped" to a chunk
- And the chunk can be accessed (via NFS) on some chunk server
  - If changed, master will later arrange to propagate the updates to other replicas that have the same chunk
  - But in any case it knows which chunk(s) are current and which server(s) have copies, so application won't see inconsistency
- Must lock files to be sure you'll see the current version
  - Locking is integrated with the master sync mechanism

23

## So…

- Conceptually, an immense but very standard file system
  - Just like on Linux or Windows…
  - Except that some files could have petabytes of data
- Only real "trick" is that to access them, you need to follow this chunk-access protocol

24

## Why Such a Large Chunk Size?

Advantages:

- Reduce clients' need to interact with the master.
- Reduce network overhead by keeping a persistent connection to the chunkserver.
- Reduce the size of metadata stored on the master.

## Why Such a Large Chunk Size?

Disadvantages:

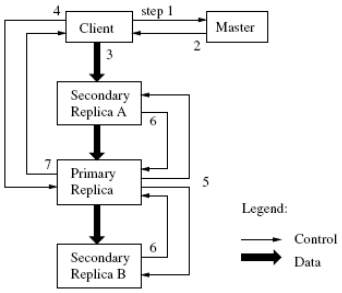
- Serious internal fragmentation when storing small files.
- Chunkservers that contain many small files may become hot spots.

## Guarantees offered by GFS

- Relaxed consistency model
- Atomic file namespace mutation with namespace locking.
- Atomic Record Append.

| | Write | Record Append |
|---|---|---|
| Serial success | *defined* | *defined* interspersed with *inconsistent* |
| Concurrent successes | *consistent* but *undefined* | |
| Failure | *inconsistent* | |

## Leases and Mutation order




## Fault-tolerance

Fast Recovery

- Master and chunkservers are designed to restore their state and start in seconds.

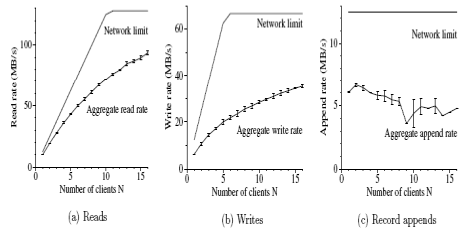
Chunk Replication

- Each chunk is replicated on mutiple chunkservers on different racks.

Master Replication

- The master state is replicated for reliability.

## Performance Measurements



## Learning more

Ghemawat, Sanjay et al. "The Google File System"

Carr, David F. "How Google Works"
http://www.baselinemag.com/c/a/Projects-Networks-and-Storage/How-Google-Works-%5B1%5D/4/

Wikipedia, "Google"
http://en.wikipedia.org/wiki/Google

## Famous file systems surprises

- Lots of gottcha's…
  - Debugging an application using printf statements
  - Then it crashes…. What was the "last line it printed"?
- Gottcha!
  - Printf is smart: collects 1024 bytes at a time and does block writes, at least normally (must call fflush() to override this behavior).
  - So the last line on the console might not be the last line it printed!

32

## Famous file systems surprises

- Process A writes to a file, then tells process B, on some other machine, to read the file
- Will process B see the file?
- Gottcha!
  - Maybe not: file I/O to the file system cache can be way ahead of the state of the disk itself
  - Must call fsync() first (and this may "hiccup" for a while)

33

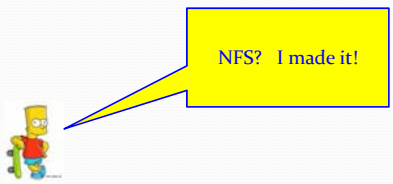
## Famous file systems surprises

- Many old Linux applications create a file called "lock" to do locking
  - Because Linux lacked a standard distributed locking mechanism
- So process A creates a file, "lock"
  - EEXISTS: A sleeps 10 seconds, then tries again
- Gottcha!
  - With remote NFS, requests are sometimes automatically reissued. A could create the lock but its operation would fail anyhow…. A then waits for itself to release the lock!

34

## Famous file systems surprises

- What did you expect?




35