CS 537
Lecture 22
Distributed File Systems

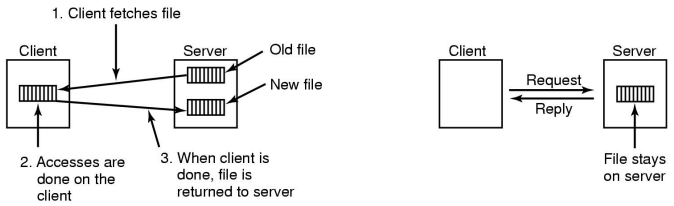
Michael Swift

Distributed File Systems

- Goal: access your data through normal file system APIs but store on some other machine
  - Sharing/collaboration with other people
  - Reliability via common backup
  - Efficient use of capacity
- 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

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
  - Provides regular behavior with multiple clients



Naming transparency

- Naming is a mapping from logical to physical objects
- Client interface may be transparent – should it?
  - 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
    - */mnt/dir1/dir2/x* says *x* is in *dir2*, but not which machine is mounted – app does not say which machine or where on the machine
  - Location independence: move files without changing names
    - Separate naming hierarchy from storage devices hierarchy

Naming Schemes

1. Files named by combination of their host name and local name; guarantees a unique system-wide name.
   - \\server\share\dir\file on Windows, or http://server/file
2. Attach remote directories to local directories, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently.
   - /mnt/dir1/dir2/file – dir1 refers to a directory on a server
3. Total integration of the component file systems.
   - A single global name structure spans all the files in the system.
   - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable.

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
- Where to cache?
  - Cliient Disk/FS: Pros: larger, data present locally on recovery
  - Client Memory: Pros: diskless workstations, quicker data access,
  - Servers: memory

Cache Update Policy

- Write-through – write data through to disk as soon as they are placed on any cache.
  - Reliable, but poor performance.
- Delayed-write – modifications written to the cache and then written through to the server later. Write accesses complete quickly; some data may be overwritten before they are written back, and so need never be written at all.
  - Poor reliability; unwritten data will be lost whenever a user machine crashes.
  - Variation – scan cache at regular intervals and flush blocks that have been modified since the last scan.
  - Variation – write-on-close, writes data back to the server . Best for files that are open for long periods and frequently modified.

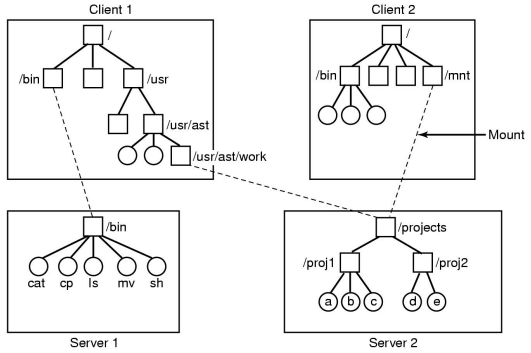
Cache Consistency

- Is locally cached copy of the data consistent with the master copy?
  - What happens if another client modifies a file you are caching?
- Client-initiated approach
  - Client initiates a validity check. (when?)
  - Server checks whether the local data are consistent with the master copy.
- Server-initiated approach
  - Server records, for each client, the (parts of) files it caches.
  - When server detects a potential inconsistency, it must reac

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
  - 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

Example



NFS

- Naming:
  - Files in NFS may have a different name on every client based on where the volume is mounted
  - A common name space can be achieved by mounting the same set of servers in the same place on every client
- Is the location transparent or location independent?

Sidebar: Remote Procedure Call

- Basic problem when dealing with machine across a network: how do you write the code to communicate?
- Option 1: messages
  - Programmer copies message into an array of bytes, “sends” to other computer, “receives” an array of bytes in response at some point
- Option 2: RPC
  - Make a procedure call that executes on the other side
  - Tool generates code to copy arguments into a message, send data, unpack data, call server code, copy result into a message, send back, receive reply, and return to caller

NFS Protocol

- Supports directory and file access via *remote procedure calls* (**RPC**s)
- 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

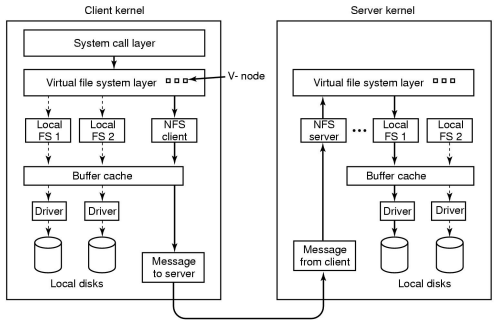
13

NFS Client 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
  - Makes RPCs for various operations to NFS server

14

NFS Layer Structure



15

Cache coherency

- Clients cache file attributes and data
  - If two clients cache the same data, cache coherency is lost
  - Modifications by one client may not be seen by the other
- Solutions:
  - Each cache block has a timer (3-30 sec)
    - 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
- Impact:
  - One client can modify data, another client may no see it for a while, but not forever.
  - New files not visible for 30 seconds

16

Andrew File System (AFS)

- Developed at CMU to support all of its student computing.
- Consists of workstation clients and **dedicated file server machines.**
- Workstations have local disks, used to cache files being used locally (originally whole files, now 64K file chunks).
- Andrew has a **single name space** -- your files have the same names everywhere in the world.

AFS Overview

- Based on the upload/download model
  - Clients download and cache files
  - Server keeps track of clients that cache the file
  - Clients upload files at end of session
- Whole file caching is central idea behind AFS
  - Download whole file first time you open it
  - Upload whole file when you modify it
- AFS servers are stateful
  - Keep track of clients that have cached files
  - Recall files that have been modified

AFS Details

- Has dedicated server machines
- Clients have partitioned name space:
  - Local name space and shared name space
  - Cluster of dedicated AFS servers present shared name space
- AFS file name works anywhere:
  - /afs/cs.wisc.edu/u/s/w/swift
  - Names are location transparent and independent
    - You don't know what server has files, nor where on that server.

AFS: Operations and Consistency

- AFS caches entire files from servers
  - Client interacts with servers only during open and close
- OS on client intercepts calls, and passes it to **AFS service** on client
- AFS service is a client process that caches files from servers
  - AFS service contacts AFS server only on open and close
    - Does not contact if file is already in the cache, and not invalidated
  - Reads and writes bypass AFS service and go right to file cached in local file system.

AFS Caching and Consistency

- Need for scaling led to reduction of client-server message traffic.
- Once a file is cached, all operations are performed locally.
  - Cache is on disk, so normal FS and FS operations work here
- On close, if the file is modified, it is replaced on the server.
  - What happens when multiple clients share a file?
- The client assumes that its cache is up to date, unless it receives a *callback* message from the server saying otherwise. On file open, if the client has received a callback on the file, it must fetch a new copy; otherwise it uses its locally-cached copy.
  - How does this compare to NFS?
- When are updates visible?
- What happens if two clients modify the file at the same time?

AFS Name lookups

- Looking up file path names is slow
  - Lots of searching directories and
- AFS offloads problem to client:
  - Client reads directory contents
    - Name, File ID (like inode number)
  - Client scans directory for name
  - Client opens file on directory by File ID
- Performance:
  - Slower than having server to lookup directly – more data back and forth
  - Supports more clients on a single server than having server do lookup

Summary

- NFS:
  - Simple distributed file system protocol. No open/close
  - Stateless server
    - Has problems with cache consistency, locking protocol
- AFS:
  - More complicated distributed file system protocol
  - Stateful server
    - session semantics: consistency on close