CS 537
Lecture 15
Distributed File Systems

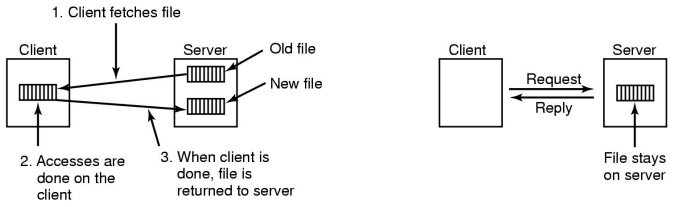
Michael Swift

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



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

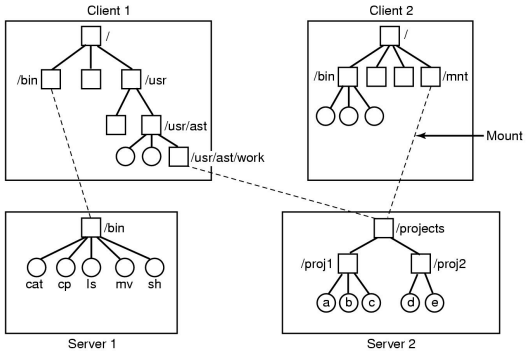
Caching

- 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

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

Example



NFS

- NFS defines a set of RPC operations for remote file access:
  - searching a directory
  - reading directory entries
  - manipulating links and directories
  - reading/writing files
- Every node may be both a client and server.

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

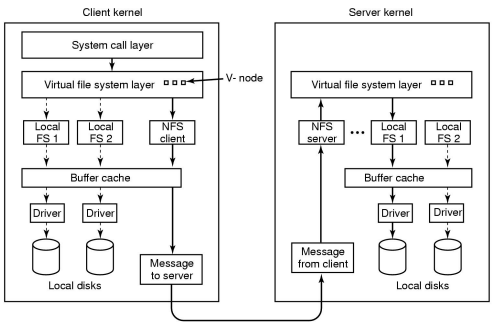
NFS Protocol

- 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

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

NFS Layer Structure



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

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.
- Andrew is good for distant operation because of its local disk caching: after a slow startup, most accesses are to local disk.

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
  - Later amended to block operations
  - Simple, effective
- 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

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 (Venus)
  - Venus is a client process that caches files from servers
  - Venus 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 Venus and go right to file cached in local file system.

17

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?

3/27/09

18

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

19