Network File Systems

- Background: what is a network file system?
- Naming
  - » Lookup
  - » Transparency
- Accessing remote files
  - » Client's view of things
  - » Server's view of things (stateful vs. stateless)
- Replicating files
  - » Performance
  - » Reliability
- Example systems

What Is a Network File System?

- Network (or distributed) file system (DFS) is
  - » File system distributed across many machines available from one or more clients
  - » File system on one machine available from many clients
- DFS may manage tens or hundreds (or more) storage devices
  - » Storage space may be broken into smaller pieces for easier management
  - » Storage spaces may be physically located in different places
- A single storage space is usually the unit that clients can choose to "import" (allow local users to access)

DFS Structures & Naming

- Server: machine that provides services to clients
  - » May be multiple servers per DFS
  - » Servers may provide different functions (naming vs. files)
- Client: process (also machine) that makes requests of servers
  - » Client interface specifies possible file operations (read, etc.)
  - » Client interface should be transparent: client process can't tell whether file is local or remote
- Naming: maps between logical names & physical objects
  - » Multilevel mapping hides the details of where the file is physically located
  - » Transparent naming in DFS hides location in network where the file is stored
  - » Name mapping may return multiple locations if there's more than one copy of the file in the DFS: this information is hidden from the client

Naming & Transparency

- Location transparency: name doesn't reveal where the file is physically stored
  - » Name still corresponds to a specific set of blocks
  - » Sharing is convenient
  - » Name translation may be easier
  - » Can cause problems if server fails or administrator wants to reorganize the FS (reallocate space)
- Location independence: name doesn't change if file changes physical location
  - » Makes it easier to share the entire storage space
  - » Separates naming issues from storage issues
  - » Makes creation of replicas easier

Approaches to Naming

- Files named by combining host name and name local to host
  - » Guarantees a unique systemwide name
  - » Causes problems if a file needs to be used
  - » Sample system: AFS
- Storage spaces (in the form of directory trees) attached to local directory tree
  - » Looks like a single directory tree
  - » Only mounted directories can be accessed
  - » Directory mount points can be changed
  - » Sample system: NFS
- Totally integrated file system
  - » Single global name space for all files and all clients
  - » Unavailable server => some files and directories may not be available

Accessing Remote Files

- Clients get files from servers when files are needed
- Clients often request the same files many times
  - » System executables
  - » User on client X always wants specific files of hers
- Reduce network traffic by keeping a copy of file blocks on the client in a *cache*
  - » Fetch data from server if data not in cache
  - » Perform accesses on cached copy
  - » Write data back to server if it changes
  - » Files still have one master copy, but may have fragments cached throughout many clients
  - » Problem: how does the DFS make sure that cached copies are *consistent* (all the same) with each other and the master file if one or more copies are written?

Caching Files: Memory or Disk?

- File data from a server may be cached on disk or in memory
- Advantages of memory:
  - » Workstations don’t need disks
  - » Memory is faster than disk
  - » Large memory can give big performance improvements
  - » Server caches are always in memory
- Advantages of disk:
  - » Cache can be larger than memory
  - » Data in cache survives reboot / failure
    - – No need to fetch after crash
    - – May be used to boot the system
  - » Data in cache is more reliable: delay writes to server longer

Caches & Writes to a File

- Write-through
  - » Data is written to the server as soon as it’s written to cache
  - » Reliable: client crash doesn’t cause lost data
  - » Poor performance: client has to write immediately
- Delayed-write
  - » Data is written to the server some time after it’s written to the client cache
    - – If file is deleted first, no write to server occurs!
    - – If data is overwritten, only one write goes to server
  - » Reliability can be low: crash causes lost data
  - » Consistency can be difficult: caches hold modified data
  - » Several variations on policy:
    - – Write after data reaches a fixed age (often ~30 seconds)
    - – Write after file has been closed: write-on-close

Keeping Data Consistent

- Clients can keep copies of the same file
- One client might write the file - how do the others find out about the change?
- Client-initiated approach
  - » Client checks with the server to see if the file has been updated before using it
  - » Server then checks to see whether any other client has written the file
- Server-initiated approach
  - » Server keeps track of which clients are caching and modifying each file
  - » Server prevents consistency, perhaps by telling clients to remove files modified elsewhere from their own caches

Stateful File Servers

- Server keeps track of which clients have opened which files, and also keep information for each file opened by a client
  - » Current position in the file
  - » Blocks the client has modified
- When client opens a file
  - » Server fetches file info from disk, holds it in memory, and gives the user an identifier for use with future accesses
  - » Server holds info in memory until file is explicitly closed
  - » Server can check security on file open and use cryptographic methods to ensure that future accesses are from the same client
- Stateful file servers can perform better
  - » Fewer security checks
  - » Server can read ahead on the file if client reads sequentially
  - » Server can keep track of multiple clients who are accessing the same file and manage consistency

Stateless File Servers

- Stateless file servers keep no per-client information
  - » Still allowed to cache inodes and file blocks in memory!
  - » Can't keep track of which files are actually open or which client is using them
- Individual requests are standalone
  - » Contain file identifier, offset in file, information on permissions
  - » File identifier need not be file name (usually inode number)
- No need for clients to open and close files
  - » Clients must still get a file identifier that corresponds to a particular file name
  - » Server has to check permissions on every request!
- Performance can be slower than stateful, but
  - » Easier to recover from client or server failure
  - » Makes maintaining consistency easier (albeit slower)

Stateful vs. Stateless File Service

- Failure recovery
  - » Stateful server loses all of its volatile state in a crash - restores state by communicating with clients
  - » Stateful server must be aware of clients that fail as well - deallocate resources used to cache their files
  - » Stateful server & client barely notice that failure has occurred
    - – Server that fails simply comes back up - no per-client information to recover
    - – Server has no info to reclaim for client that fails
- Consistency
  - » Stateless servers can't easily maintain consistency themselves
  - » Stateful servers can track who's caching which files
- Performance
  - » Stateful servers tend to be faster
  - » Stateless servers recover from failures faster

Replicating Files

- Problem: file server crashes => file unavailable (or even lost!)
- Solution: keep multiple copies of the file on different servers
- Benefits
  - » Improves availability & reliability
  - » May improve performance (get the nearest copy)
- Issues:
  - » Naming scheme must map name to a particular replica
    - – Pick the nearest or least loaded server
    - – Existence of replicas must be invisible to clients
  - » Consistency
    - – Updates must go to all replicas
    - – Consistency must be kept as if all replicas were a single file

NFS

- NFS (Network File System) is a classic distributed file system
- Naming
  - » Names are location transparent but not location independent
  - » Names need not be consistent between two clients
- Server state
  - » Stateless file servers: easier to recover from crashes (which were relatively common when NFS was designed)
  - » Each request must contain all information necessary for the I/O, including user & authentication info
- Replication
  - » No automatic replication
  - » Clients can keep copies in their local file systems
- Security
  - » Hah!

AFS

- AFS (Andrew File System)
- Naming
  - » Domains are mentioned as part of the name
  - » Names within a domain are location transparent & independent
- Server state
  - » Stateful file servers - slower & more complex recovery, but better performance
  - » Authentication done only when the file is opened
- Replication
  - » Automatic replication is supported
  - » Clients can keep copies long-term locally, particularly if they don't change often (system files)
- Security
  - » Pretty good (uses Kerberos)