Distributed Storage

March 12, 2001

What is Distributed Storage?

- Simple answer: Storage that can be shared throughout a network
- Simple examples: Windows file sharing, ftp
- More sophisticated examples: NFS, AFS, distributed shared memory
- Key Objectives: speed, availability, consistency, fault tolerance, security, transparency

Distributed File Storage

- Typically dedicated file servers
- Network and disk subsystems highly optimized
  - RAIDs, large file caches, high-performance Ethernet cards
- File servers manage the bits
  - Files can be replicated for speed, availability, and some fault tolerance
  - If replicated, consistency can be difficult depending on level of transparency

Distributed File Storage Issues

- Security - Can be built in at different layers
  - to protect files, can use passwords, tickets, access control lists, encryption, ...
- Fault Tolerance - Servers can go down at any time
  - Can make servers "stateless"
  - Can use transactions or logging

NFS

- Developed by SUN
- Based on RPC
- Can be implemented on top of UDP or TCP
- Transparent - Virtual File System

NFS, cont.

- Files identified by File Handles
  - System calls use file handles (e.g., lookup, link, statfs)
  - File Handle contains a file i-node, an i-node generation number, and a file system id
- i-node generation number - needed because i-nodes are reused

NFS Mounting

- Local file systems that can be mounted remotely stored in /etc/exports
- Hard mounts - file system access is a blocking operation
- Soft mounts - file system access is a non-blocking operation
  - Less Transparency
  - Allows client to determine what happens when a failure occurs
  - Not backwards compatible!

NFS Performance Improvements

- Locally, typically use read-ahead, write-behind
- NFS Servers can use read-ahead as-is, but write caching raises fault tolerance issues
  - e.g., what if the server stores a write in the cache, and then dies?
- Option 1: write-through caching
  - Problem: Performance bottleneck!
- Option 2: Writes are cached, client determines when cache is written

Availability

- Replication makes files more available, but introduces consistency issues
- NFS supports replication of read-only files, but not of read/write files
- One-copy semantics + replication does not scale easily
- Instead, need to make assumptions

AFS

- Each file has a custodian responsible for it (e.g., a file server)
- Clients "check out" a copy of a file
  - Client caches file locally, like any other file
- When the file is updated back to custodian, Custodian notifies all clients who checked out a file
  - Client must ensure that it has a fresh copy at least once every T second (system parameter)