8 Distributed File Systems

8.1 Main Points

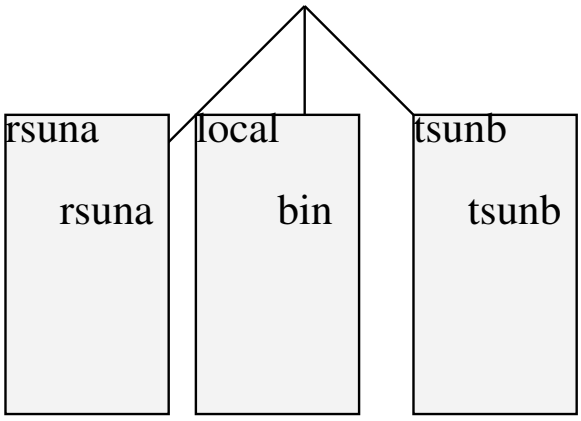
A **Distributed File System** provides transparent access to files stored on a remote disk

Themes:

- Failures - what happens when server crashes but client doesn't? or vice versa
- Performance ⇒ Caching - use caching at both the clients and the server to improve performance
- Cache Coherence - how do we make sure each client sees most up to date copy of file?

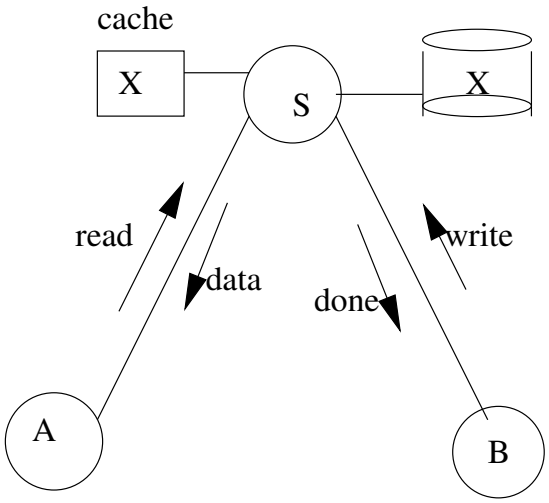
8.2 Client Implementation

- Request for operation on file goes to OS.
- OS recognises file is remote and constructs RPC to remote file server
- Server receives call, does operation and returns results.
- Subtrees in directory structure generally map to file system. Provides access transparency



8.3 No Caching

- Simple approach: Use RPC to forward each file system request to remote server (older versions of Novell Netware).
- Example operations: `open, seek, read, write, close`
- Server implements operations as it would for local request and passes result back to client



8.3.1 Advantages and Disadvantages of uncached remote file service

Advantage server provides consistent view of file system to both A and B.

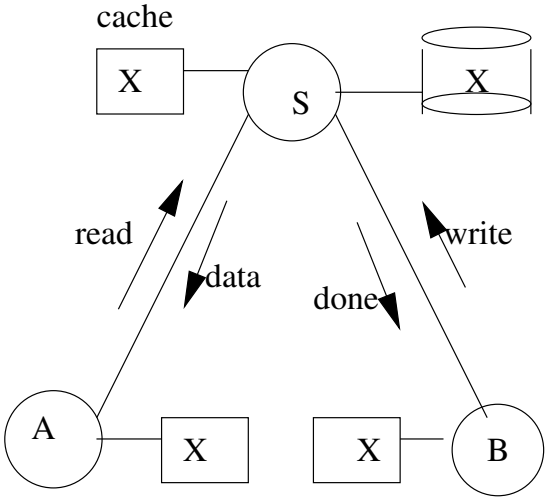
Disadvantage can be lousy performance

- Going over network is slower than going through memory
- Lots of network traffic - congestion
- Server can be a bottleneck - what if lots of clients

8.4 NFS - Sun Network File System

Main idea - uses caching to reduce network load

- Cache file blocks, file headers etc in both client and servers memory.
- More recent NFS implementations use a disk cache at the client as well.



Advantage Advantage: If open/read/write/close can be done locally, no network traffic

Issues: failure and cache consistency

8.4.1 Failures

What if server crashes? Can client wait until server comes back up and continue as before?

1. Any data in server memory but not yet on disk can be lost
2. If there is shared state across RPCs, eg open seek read. What if server crashes after seek? Then when client does "read" it will fail.
3. Message re-tries - suppose server crashes after it does "rm foo" but before it sends acknowledgement? Message system will retry and send it again. How does system know not to delete it again (someone else may have created new "foo"). Could use transactions, but NFS takes more ad hoc approach.

What if client crashes?

1. Might lose modified data in client cache.

8.4.2 NFS Protocol

1. Write-through caching - when a file is modified, all modified blocks are sent immediately to the server disk. To the client, "write" doesn't return until all bytes are stored on disk.
2. Stateless Protocol - server keeps no state about client, except as hints to improve performance
   - Each read request gives enough information to do entire operation - `ReadAt(inumber, position)` not `Read(openFile)`.
   - When server crashes and restarts, can start processing requests immediately, as if nothing had happened.
3. Operations are *idempotent*: all requests are ok to repeat (all requests are done at least once). So if server crashes between disk I/O and message send, client can resend message and server does request over again
   - Read and write file blocks are easy - just re-read or re-write file block, so no side-effects.
   - What about "remove"? NFS just ignores this - does the remove twice and returns error if file not found, application breaks if inadvertently removes other file.
4. Failures are transparent to client system.
   Is this good idea? What should happen if server crashes? If application in middle of reading file when server crashes, options:
   (a) Hang until server returns (next week...)
   (b) return an error? But networked file service is transparent. Application doesn't know network is there. Many Unix Apps ignore errors and crash if there is a problem.

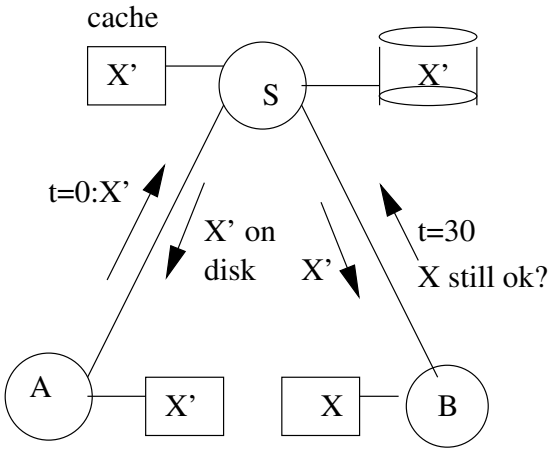
   NFS has both options - the administrator can select which one. Usually hang and return error if absolutely necessary.

8.4.3 Cache consistency

- What if multiple clients sharing same files? Easy if they are both reading - each gets a local copy in their cache.
- What if one writing? How do updates happen?
- Note NFS has write-through cache policy. If one client modifies file, writes through to server.
- How does other client find out about change?

8.4.4 NFS and weak consistency

- In NFS, client polls server periodically to check if file has changed. Polls server if data hasn't been checked in every 3-30 seconds (Exact time is tunable parameter)



- When file changed on one client, server is notified, but other clients use old version of file till timeout. They then check server and get new version.
- What if multiple clients write to same file? In NFS get either version or mixed version. Completely arbitrary!

8.4.5 Sequential ordering constraints

- What should happen? If one CPU changes file, and before it completes, another CPU reads file?
- We want isolation between operations, so read should get old file if it completes before write starts, new version if it starts after write completes. Either all new or all old any other way cf serialisability.

8.4.6 NFS Pros and Cons

- its simple
- its highly portable
- *but* its sometimes inconsistent
- *but* it doesn't scale to large numbers of clients

Note that this describes NFS version 3.

8.5 Andrew File System

AFS (CMU late 80s) ⇒ DCE DFS (commercial product)

1. Callbacks: Server records who has copies of file
2. Write through on close
   - If file changes, server is updated (on close)
   - Server then immediately tells all those with old copy

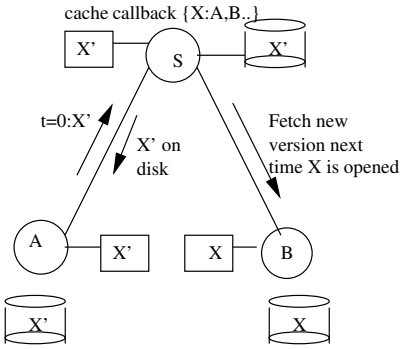
8.5.1 AFS Session Semantics

Session semantics - updates only visible on close

- In Unix (single machine), updates visible immediately to other programs who have file open.
- In AFS, everyone who has file sees old version, anyone who opens file again will see new version.

  In AFS:

  1. on open and cache miss: get file from server, set up callback
  2. on write close: send copy to server; tells all clients with copies to fetch new version from server on next open
- Files cached on local disk; NFS (used to) cache only in memory



- What if server crashes? Lose all your callback state.
- Reconstruct information from client - go ask everyone "who has which files cached"

8.5.2 AFS Pros and Cons

- Disk as cache ⇒ more files cached locally
- Callbacks ⇒ server not involved if file is read-only (Majority of file access is read-only)
- But on fast LANs, local disk is slower than remote memory

NFS version 4 will provide *session semantics* when it is deployed by vendors in the 2005 timeframe.

8.6 Summary

- Remote file performance needs caching to get decent performance.
- Central server is a bottleneck
  - Performance bottleneck:
    - All data is written through to server
    - all cache misses go to server
  - Availability Bottleneck
    - Server is single point of failure
  - Cost bottleneck
    - Server machines high cost relative to workstation