Network File System (NFS)

Nima Honarmand
(Based on slides by Don Porter and Mike Ferdman)

**Stateful vs. Stateless Protocols**

- **Stateful protocol:** server keeps track of past requests
  - i.e., state persist across requests on the server
- **Stateless protocol:** server does not keep track of past requests
  - Client should send all necessary state with a single request
- Challenge of stateful: Recovery from crash/disconnect
  - Server side challenges:
    - Knowing when a connection has failed (timeout)
    - Tracking state that needs to be cleaned up on a failure
  - Client side challenges:
    - If server thinks we failed (timeout), must recreate server state

**Stateful vs. Stateless Protocols**

- Drawbacks of stateless:
  - May introduce more complicated messages
  - And more messages in general

**NFS is Stateless**

- Every request sends all needed info
  - User credentials (for security checking)
  - File handle and offset
- Each request matches a VFS operation
  - e.g., lookup, read, write, unlink, stat
  - there is no open or close among NFS operations
- Default NFS transport protocol (up to NFSv3) was UDP.

**Intuition and Challenges**

**Intuition:**
- Translate VFS requests into remote procedure calls to server
  - Instead of translating them into disk accesses

**Challenges:**
- Server can crash or be disconnected
- Client can crash or be disconnected
- How to coordinate multiple clients on same file?
- Security
  - ...

From Sandberg et al., 1985
### Challenge: Lost Request?
- Request sent to NFS server, no response received
  - Did the message get lost in the network (UDP)?
  - Did the server die?
  - Is the server slow?
  - Don’t want to do things twice
    - Bad idea: write data at the end of a file twice
- Idea: Make all requests idempotent
  - Requests have same effect when executed multiple times
    - Ex: `write()` has an explicit offset, same effect if done twice
  - Some requests not easy to make idempotent
    - E.g., deleting a file
  - Server keeps a cache of recent requests and ignores requests found in the cache

### Challenge: inode Reuse
- Process A opens file ‘foo’
  - Maps to inode 30
- Process B unlinks file ‘foo’
  - On local system, OS holds reference to the inode alive
  - NFS is stateless, server doesn’t know about open handle
    - The file can be deleted and the inode reused
    - Next request for inode 30 will go to the wrong file
- Idea: Generation Numbers
  - If inode in NFS is recycled, generation number is incremented
  - Client requests include an inode + generation number
    - Enables detecting attempts to access an old inode

### Challenge: File Locking
- Must have way to change file without interference
  - Get a server-side lock
    - What happens if the client dies?
    - Lots of options (timeouts, etc), mostly bad
    - Pointed to a separate, optional locking service
      - Such as Network Lock Manager (NLM)
      - With ugly hacks and timeouts

### Challenge: Removal of Open Files
- Recall: Unix allows accessing deleted files if still open
  - Reference in in-memory inode prevents cleanup
    - Applications expect this behavior; how to deal with it in NFS?
  - On client, check if file is open before removing it
    - If yes, rename file instead of deleting it
    - *id* files in modern NFS
  - When file is closed, delete temp file
    - If client crashes, garbage file is left over
    - Only works if the same client opens and then removes file

### Challenge: Security
- Local UID/GID passed as part of the call
  - UIDs must match across systems
    - Yellow pages (yp) service; evolved to NIS
    - Replaced with LDAP or Active Directory
- Problem with "root": root on one machine becomes root everywhere
  - Solution: root squashing – root (UID 0) mapped to "nobody"
    - Ineffective security
      - Can send any UID in the NFS packet
      - With root access on NFS client, "su" to another user to get UID
    - Unlikely, but possible
      - DirectoryEntry: one way of implementing "nobody"
      - But still possible to do harmful things

### Challenge: Time Synchronization
- Each CPU’s clock ticks at slightly different rates
  - These clocks can drift over time
- Tools like ‘make’ use timestamps
  - Clock drift can cause programs to misbehave
    - `make`: warning: Clock skew detected.
    - `make`: warning: Clock skew detected.
    - Your build may be incomplete.
- Systems using NFS must have clocks synchronized
  - Using external protocol like Network Time Protocol (NTP)
    - Synchronization depends on unknown communication delay
    - Very complex protocol but works pretty well in practice
Challenge: Caches and Consistency
• Clients A and B have file in their cache
• Client A writes to the file
  – Data stays in A’s cache
  – Eventually flushed to the server
• Client B reads the file
  – Does B see the old contents or the new file contents?
    • Who tells B that the cache is stale?
    • Server can tell, but only after A actually wrote/flushed the data

Consistency/Performance Tradeoff
• Performance: cache always, write when convenient
  – Other clients can see old data, or make conflicting updates
• Consistency: write everything immediately
  – And tell everyone who may have it cached
    • Requires server to know the clients which cache the file (stateful??)
  – Much more network traffic, lower performance
  – Not good for the common case: accessing an unshared file

Close-to-Open Consistency
• NFS Model: Flush all writes on a close
  – If stale, invalidate the cache
  – Makes sure you get the latest version on the server when opening a file

NFS Evolution
• The simple protocol was version 2
• Version 3 (1995):
  – 64-bit file sizes and offsets (large file support)
  – Bundle attributes with other requests to eliminate stat()
  – Other optimizations
  – Still widely used today

NFSv4 (2000)
• Attempts to address many of the problems of v3
  – Security (eliminate homogeneous UID assumptions)
  – Performance
• Provides a stateful protocol
• pNFS – extensions for parallel distributed accesses
• Too advanced for its own good
  – Much more complicated than v3
  • Slow adoption
  – Barely being phased in now
    • With hacks that lose some of the features (looks more like v3)