Distributed File Systems
Case Studies
NFS
Network File System
Sun Microsystems

c. 1985
NFS Design Goals

- Any machine can be a **client** or **server**
- Must support **diskless workstations**
- **Heterogeneous systems** must be supported
  - Different HW, OS, underlying file system
- **Access transparency**
  - Remote files accessed as local files through normal file system calls (via VFS in UNIX)
- **Recovery from failure**
  - Stateless, UDP, client retries
- **High Performance**
  - use caching and read-ahead
NFS Design Goals

No migration transparency

If resource moves to another server, client must remount resource.
NFS Design Goals

No support for UNIX file access semantics
Stateless design: file locking is a problem.

All UNIX file system controls may not be available.
NFS Design Goals

Devices

**must** support diskless workstations where *every* file is remote.

Remote devices refer back to local devices.
NFS Design Goals

Transport Protocol
Initially NFS ran over **UDP** using Sun RPC

Why UDP?
- Slightly faster than TCP
- No connection to maintain *(or lose)*
- Designed for ethernet LAN environment
  relatively reliable

- Error detection but no correction.
- NFS retries requests

---

Paul Krzyzanowski • Distributed Systems
NFS Protocols

- Mounting protocol
  - Request access to exported directory tree

- Directory & File access protocol
  - Access files and directories (read, write, ...)

Paul Krzyzanowski • Distributed Systems
Mounting Protocol

- Send pathname to server
- Request permission to access contents

**client**: parses pathname
contacts server for file handle

- Server returns **file handle**
  - File device #, inode #, instance #

**client**: create in-code vnode at mount point.
(points to inode for local files)
points to **rnode** for remote files
  - stores state on client
Mounting Protocol

- **static mounting**
  - Mount request contacts server

Server: /etc/exports
Client: mount fluffy:/users/paul /home/paul
Directory and file access protocol

- Initially perform *lookup* RPC
  - returns *file handle* and attributes

- *Not like open*
  - No information is stored on server

- handle passed as a parameter for other file access functions
  - e.g. `read(handle, offset, count)`
Directory and file access protocol

- NFS has 16 functions
  - (version 2; six more added in version 3)

  - null
  - lookup
  - create
  - remove
  - rename
  - read
  - write
  - link
  - symlink
  - readdir
  - mkdir
  - rmdir
  - readdir
  - getattr
  - setattr
  - statfs
Accessing files

- Parse **component at a time** via *namei*
  - At each point, see if mount point
    - Yes? Continue on the mounted file system
    - Remote? Perform NFS RPC *lookup*
- Ensures that .. is processed locally and future mount points are processed
- Final lookup returns handle
- Create in-core vnode, rnode
Accessing files

Application can now access file

file descriptor → in-core vnode (VFS layer)

↑

in-core rnode (NFS client)

Perform NFS read/write RPCs using state in rnode

RPCs include user ID and group ID
- security hole
**NFS Performance**

- Usually slower than local
- Improve by caching at client
  - Goal: reduce number of remote operations
  - Cache results of `read, readlink, getattr, lookup, readdir`
  - Cache file data at client (buffer cache)
  - Cache file attribute information at client
  - Cache pathname bindings for faster lookups
- Server side
  - Caching is “automatic” via buffer cache
  - All NFS writes are *write-through* to avoid unexpected data loss if server dies
Inconsistencies may arise

- Try to resolve by **validation**
  - Save timestamp of file
  - When file opened or server contacted for new block
    - Compare last modification time
    - If remote is more recent, invalidate cached data
Validation

- Always invalidate data after some time
  - After 3 seconds for open files (data blocks)
  - After 30 seconds for directories

- If block is modified
  - Marked *dirty*
  - Scheduled to be written
  - Flushed on close
NFS read-ahead

- Transfer data in large chunks
  - 8K bytes default

- As soon as a chunk is received
  - A new read request is issued for the next chunk
  - Assumes data is read in-order
NFS read-ahead

application | kernel | server
---|---|---
\text{read}(\text{byte 0}) & \text{request bytes 0..8191} & \text{return bytes 0..8191} \\
\text{wait...} & \text{read}(\text{byte 0}) & \text{return bytes 0..8191} \\
\text{return}(\text{byte 0}) & \text{read}(\text{byte 1}) & \text{return bytes 0..8191} \\
\text{return}(\text{byte 1}) & \text{read}(\text{byte 8191}) & \text{return bytes 0..8191} \\
\text{return}(\text{byte 8191}) & \text{read}(\text{byte 8192}) & \text{request bytes 8192..16535} \\
\text{wait...} & \text{return}(\text{byte 8192}) & \text{return bytes 8192..16535}
Problems with NFS

- File consistency
- Assumes clocks are synchronized
- Open with append cannot be guaranteed to work
- Locking cannot work
  - Separate lock manager added (stateful)
- No reference counting of open files
  - You can delete a file you (or others) have open!
- Global UID space assumed
Problems with NFS

• No reference counting of open files
  – You can delete a file you (or others) have open!

• Common practice
  – Create temp file, delete it, continue access
  – Sun’s hack:
    • If same process with open file tries to delete it
    • Move to temp name
    • Delete on close
Problems with NFS

- File permissions may change
  - Invalidating access to file
- No encryption
  - Requests via unencrypted RPC
  - Authentication methods available
    - Diffie-Hellman, Kerberos, Unix-style
  - Rely on user-level software to encrypt
Improving NFS: version 2

- user-level lock manager
- NV RAM support
  - Improves write performance
  - Normally NFS must write to disk on server before responding to client write requests
  - Relax this rule through the use of non-volatile RAM
Improving NFS: version 2

• Adjust RPC retries dynamically
  – Reduce network congestion from excess RPC retransmissions under load
  – Based on performance

• Client-side disk caching
  – cacheFS
  – Extend buffer cache to disk for NFS
    • Cache in memory first
    • Cache on disk in 64KB chunks
Improving NFS: version 2

- Enhanced lock manager
  - Monitored locks
    - status monitor: monitors hosts with locks
    - Informs lock manager if host inaccessible
    - If server crashes: status monitor reinstates locks on recovery
    - If client crashes: all locks from client are freed
The automounter

- Problem with mounts
  - If a client has many remote resources mounted, boot-time can be excessive
  - Each machine has to maintain its own name space
    - Painful to administer on a large scale

- Automounter
  - Allows administrators to create a global name space
  - Support *on-demand* mounting
Automounter

- Solve static mounting problems with... the automounter
- Mount and unmount in response to client demand
  - Set of directories are associated with a local directory
  - None are mounted initially
  - When local directory is referenced
    - OS sends a message to each server
    - First reply wins
  - Attempt to unmount every 5 minutes
Automounter maps

- Automounter maps
  - Provide mapping: client pathname → server file system
  - Automounter converts maps into mounts that are added to the client’s file system tree

- Alternative to automounter maps
  - X/Open Federated Naming Specification (XFN)
  - Name service
  - All resources under /xfn/pathname
Automounter maps

Example:

    automount /usr/src srcmap

srcmap contains:

<table>
<thead>
<tr>
<th>cmd</th>
<th>-ro</th>
<th>doc:/usr/src/cmd</th>
</tr>
</thead>
<tbody>
<tr>
<td>kernel</td>
<td>-rw</td>
<td>frodo:/release/src \</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bilbo:/library/source/kernel</td>
</tr>
<tr>
<td>lib</td>
<td>-ro</td>
<td>sneezy:/usr/local/lib</td>
</tr>
</tbody>
</table>

Access /usr/src/cmd: request goes to doc
Access /usr/src/kernel:
    ping frodo and bilbo, mount first response

Paul Krzyzanowski • Distributed Systems
Inside the automounter

- automounter speaks NFS and mount protocols
- Performs NFS mount of given directory
  - Mount has address and port number of automounter, *not* nbsd server
- All NFS requests go to automounter
- When automounter gets a *lookup* RPC of a directory in the map
  - Mounts the directory under a directory in /tmp_mnt
  - Returns a symbolic link to the directory

Paul Krzyzanowski • Distributed Systems
The automounter

application → automounter → server

NFS mount

NFS request

KERNEL
VFS
NFS

NFS request

Paul Krzyzanowski • Distributed Systems
More improvements… NFS v3

- New version of NFS protocol
- Support 64-bit file sizes
- TCP support and large-block transfers
  - UDP caused more problems on WANs (errors)
  - All traffic can be multiplexed on one connection
    - Minimizes connection setup
  - No fixed limit on amount of data that can be transferred between client and server
- Server checks access for entire path from client
More improvements… NFS v3

- Negotiate for optimal transfer size
- New *commit* operation
  - Check with server after a *write* operation to see if data is committed
  - If *commit* fails, client must *resend* data
  - Reduce number of *write* requests to server
  - Speeds up *write* requests
    - Don’t require server to write to disk immediately
- Return file attributes with each request
  - Saves extra RPCs
SMB
Server Message Blocks
Microsoft
c. 1987
SMB Goals

- File sharing protocol for Windows 95/98/NT/2000/ME/XP
- Protocol for sharing
  - Files, devices, communication abstractions (named pipes), mailboxes
- Servers: make file system and other resources available to clients
- Clients: access shared file systems, printers, etc. from servers

**Design Priority:**
locking and consistency over client caching
**SMB Design**

- Request-response protocol
  - Send and receive *message blocks*
    - name from old DOS system call structure
  - Send *request* to server (machine with resource)
  - Server sends response

- Connection-oriented protocol

- Each message contains:
  - Fixed-size header
  - Command string (based on message) or reply string
**Message Block**

- **Header:** [fixed size]
  - Protocol ID
  - Command code (0..FF)
  - Error class, error code
  - Tree ID – unique ID for resource in use by client (handle)
  - Caller process ID
  - User ID
  - Multiplex ID (to route requests in a process)
- **Command:** [variable size]
  - Param count, params, #bytes data, data
SMB Commands

- Files
  - Get disk attr
  - create/delete directories
  - search for file(s)
  - create/delete/ rename file
  - lock/unlock file area
  - open/commit/close file
  - get/set file attributes
SMB Commands

• Print-related
  – Open/close spool file
  – write to spool
  – Query print queue

• User-related
  – Discover home system for user
  – Send message to user
  – Broadcast to all users
  – Receive messages
Protocol Steps

- Establish connection
- Negotiate protocol

- Authenticate/set session parameters
  - Send `sesssetupX` SMB with username, password
  - Receive NACK or UID of logged-on user
  - UID must be submitted in future requests
Protocol Steps

- Establish connection
- Negotiate protocol - negprot
- Authenticate - sesssetupX
- Make a connection to a resource
  - Send tcon (tree connect) SMB with name of shared resource
  - Server responds with a tree ID (TID) that the client will use in future requests for the resource
Protocol Steps

- Establish connection
- Negotiate protocol - negprot
- Authenticate - sesssetupX
- Make a connection to a resource – tcon
- Send open/read/write/close/... SMBs
Locating Services

- Clients can be configured to know about servers
- Each server broadcasts info about its presence
  - Clients listen for broadcast
  - Build list of servers
- Fine on a LAN environment
  - Does not scale to WANs
  - Microsoft introduced *browse servers* and the *Windows Internet Name Service (WINS)*
Security

- Share level
  - Protection per “share” (resource)
  - Each share can have password
  - Client needs password to access all files in share
  - Only security model in early versions
  - Default in Windows 95/98

- User level
  - Protection applied to individual files in each share based on access rights
  - Client must login to server and be authenticated
  - Client gets a UID which must be presented for future accesses

Paul Krzyzanowski • Distributed Systems
**SMB evolves**

SMB reverse-engineered
- **samba** under Linux

Microsoft released protocol to X/Open in 1992

Microsoft, Compaq, SCO, others joined to develop an enhanced public version of the SMB protocol:

**Common Internet File System (CIFS)**
Goals

- Heterogeneous HW/OS to request file services over network
- Based on SMB protocol
- Support
  - Shared files
  - Byte-range locking
  - Coherent caching
  - Change notification
  - Replicated storage
  - Unicode file names
Goals

- Applications can register to be notified when file or directory contents are modified
- Replicated virtual volumes
  - For load sharing
  - Appear as one volume server to client
  - Components can be moved to different servers without name change
  - Use referrals
  - Similar to AFS
Goals

- Batch multiple requests to minimize round-trip latencies
  - Support wide-area networks
- Transport independent
  - But need reliable connection-oriented message stream transport
- DFS support (compatibility)
Caching and Server Communication

- Increase effective performance with
  - Caching
    - Safe if multiple clients reading, nobody writing
  - read-ahead
    - Safe if multiple clients reading, nobody writing
  - write-behind
    - Safe if only one client is accessing file

- Minimize times client informs server of changes
Oplocks

Server grants **opportunistic locks** (oplocks) to client
- Oplock tells client how/if it may cache data
- Enhancement of DFS tokens

Client must request an oplock
- oplock may be
  - Granted
  - Revoked
  - Changed by server

Paul Krzyzanowski • Distributed Systems
Level 1 oplock

- Client can open file for exclusive access
- Arbitrary caching
- Cache lock information
- Read-ahead
- Write-behind

If another client opens the file, the server has former client **break its oplock:**

- Client must send server any lock and write data and acknowledge that it does not have the lock
- Purge any read-aheads
Level 2 oplock

- Request if expect others to read
- Multiple clients may have the same file open as long as none are writing
- Cache reads, file attributes
- Send other requests to server

Level 2 oplock revoked if another client opens the file for writing
Batch oplock

- Client can keep file open on server even if a local process that was using it has closed the file
- Client requests batch oplock if it expects programs may behave in a way that generates a lot of traffic (e.g. accessing the same files over and over)
  - Designed for Windows batch files

Batch oplock revoked if another client opens the file
Filter oplock

- Open file for read or write
- Locks file so other clients cannot open for write or delete
  - All clients can share read access
- Allow other clients to perform non-intrusive (read) operations
No oplock

- All requests must be sent to the server

- can work from cache *only* if byte range was locked by client
CIFS Summary

• Standard has not yet materialized
  – Future uncertain
• Oplocks mechanism supported in Windows NT, 2000, XP
• Oplocks offer flexible control for distributed consistency