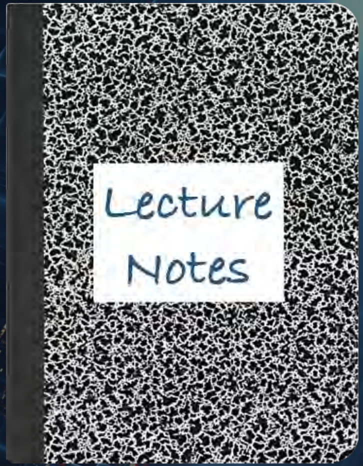


CS 417 – DISTRIBUTED SYSTEMS

Week 6: Distributed File Systems

Part 2: NFS

Paul Krzyzanowski

NFS

Network File System

Sun Microsystems

NFS Design Goals

- Any machine can be a client or server
- Must support diskless workstations
  - Device files refer back to local drivers
- Heterogeneous systems
  - Not 100% for all UNIX system call options
- Access transparency: normal file system calls
- Recovery from failure:
  - Stateless, UDP, client retries
  - Stateless → no locking!
- High Performance
  - use caching and read-ahead

NFS Design Goals

Transport Protocol

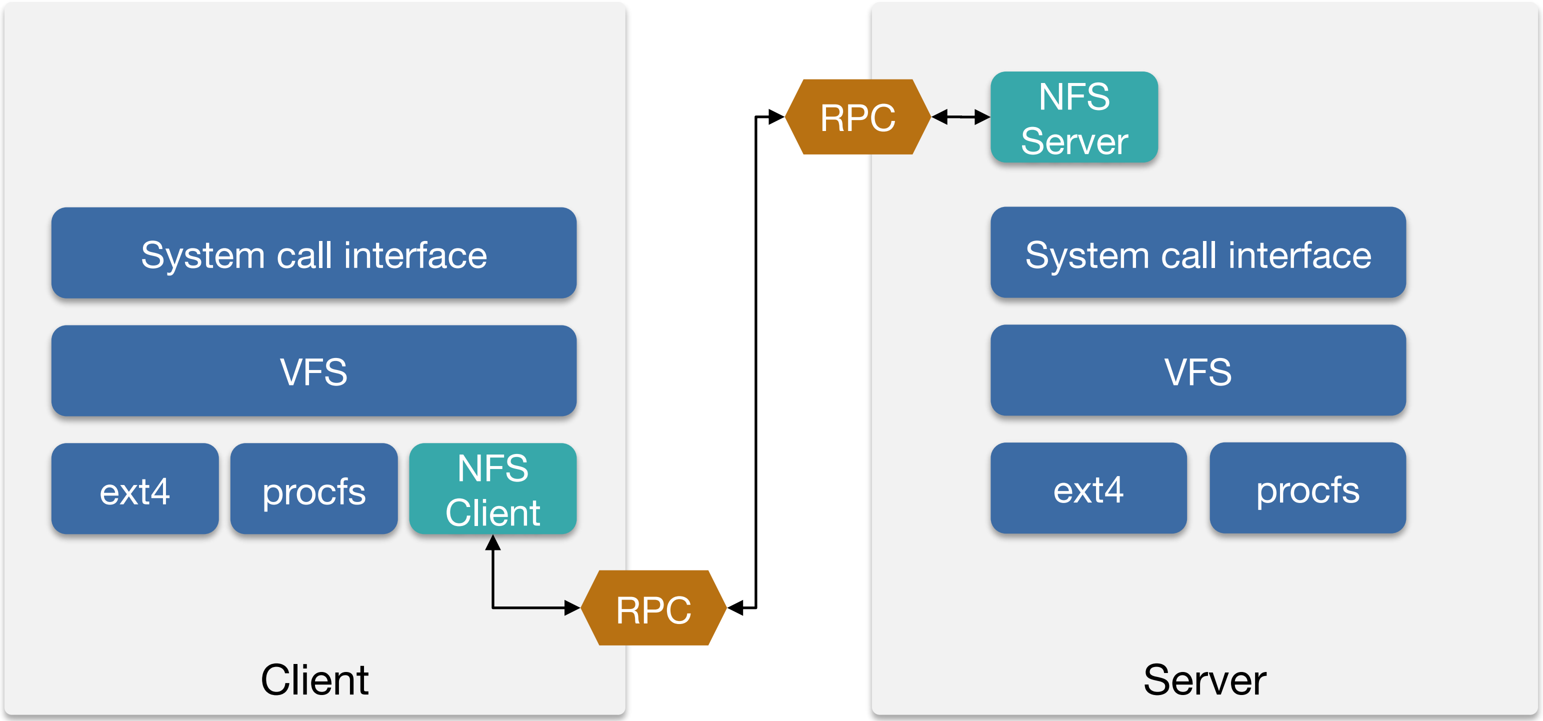
Initially NFS ran over UDP

Requests used Sun (ONC) RPC

Why was UDP chosen?

- Slightly faster than TCP
- No connection to maintain (*or to lose*)
- NFS is designed for Ethernet LAN environment – relatively reliable
- UDP has error detection (drops bad packets) but no retransmission (the RPC system will retry RPCs with no responses)

VFS on client; Server accesses local file system



NFS Protocols

Mounting protocol

Request access to exported directory tree

Directory & File access protocol

Access files and directories
(*read, write, mkdir, readdir*, … operations)

Mounting Protocol

```
mount fluffy:/users/paul /home/paul
```

- Send pathname to server
  - Request permission to access contents

client: parses pathname
contacts server for file handle

- Server validates access
  - Requested pathname must be in the file `/etc/exports`
  - Returns file handle = file device #, inode #, instance #

client: create in-memory VFS ***inode*** at mount point
internally points to ***rnode*** the NFS driver to track remote file systems
- *Client keeps state, not the server*

Directory and file access protocol

- First, perform a *lookup* RPC
  - returns file handle and attributes
  - lookup is ***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
readlink

mkdir
rmdir
readdir

getattr
setattr

statfs

Improving NFS Performance

- Usually slower than local

- Improve by caching at client

  Goal: reduce need for 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 disk to avoid unexpected data loss if server dies

Improving NFS *read* performance

- Transfer data in **chunks**
  - 8K bytes default

- **Read-ahead**
  - Optimize for sequential file access
  - Send requests to read disk blocks before they are requested by the application

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

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

- If a data block is modified, it is:
  - Marked *dirty*
  - Scheduled to be written → **Not sent to the server immediately!**
  - Flushed on file close

Problems with NFS

- File consistency
- Assumes clocks are synchronized
- Open with append cannot be guaranteed to work
  - *getattr* & *write(offset)* are separate operations
- Locking cannot work
  - Separate lock manager added (but this adds stateful behavior)
- No reference counting of open files at the server
  - You can delete a file that you (or others) have open!

- File permissions may change
  - Invalidating access to file
- Global UID space assumed
- No encryption or authentication
  - Requests via unencrypted RPC
  - Authentication methods were later added:
    - Diffie-Hellman, Kerberos, Unix-style
  - Rely on user-level software to dataencrypt

Early NFS enhancements (v2)

- **User-level lock manager**
  - Monitored locks: introduces *state* at server (but runs as a separate user-level process)
    - Status monitor: monitors clients 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
- **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

Early NFS enhancements (v2)

- **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

More improvements… NFS v3

- Updated 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
- Negotiate for optimal transfer size

More improvements… NFS v3

- 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 to get attributes for validation

The End