ADVANCED OPERATING SYSTEMS

NFS

AFS

Design and implementation of the Sun Network Filesystem (1985)

- What kind of paper is this?
  - New big idea?
  - Measurement paper?
  - Experiences/lessons learnt paper?
  - A system description?
  - Performance study?
  - Refute-conventional wisdom?
  - Survey paper?

What kind of paper is this?

- Motivate need for system
- Establish goals
- Describe real system
- Evaluate performance
- Design modifications into system; not glued on the side

Is NFS a file system?

- What is a file system?
- Is NFS a file system?

Is NFS a file system?

- What is a file system?
- Is NFS a file system?
  - NFS is a remote access protocol

Goals

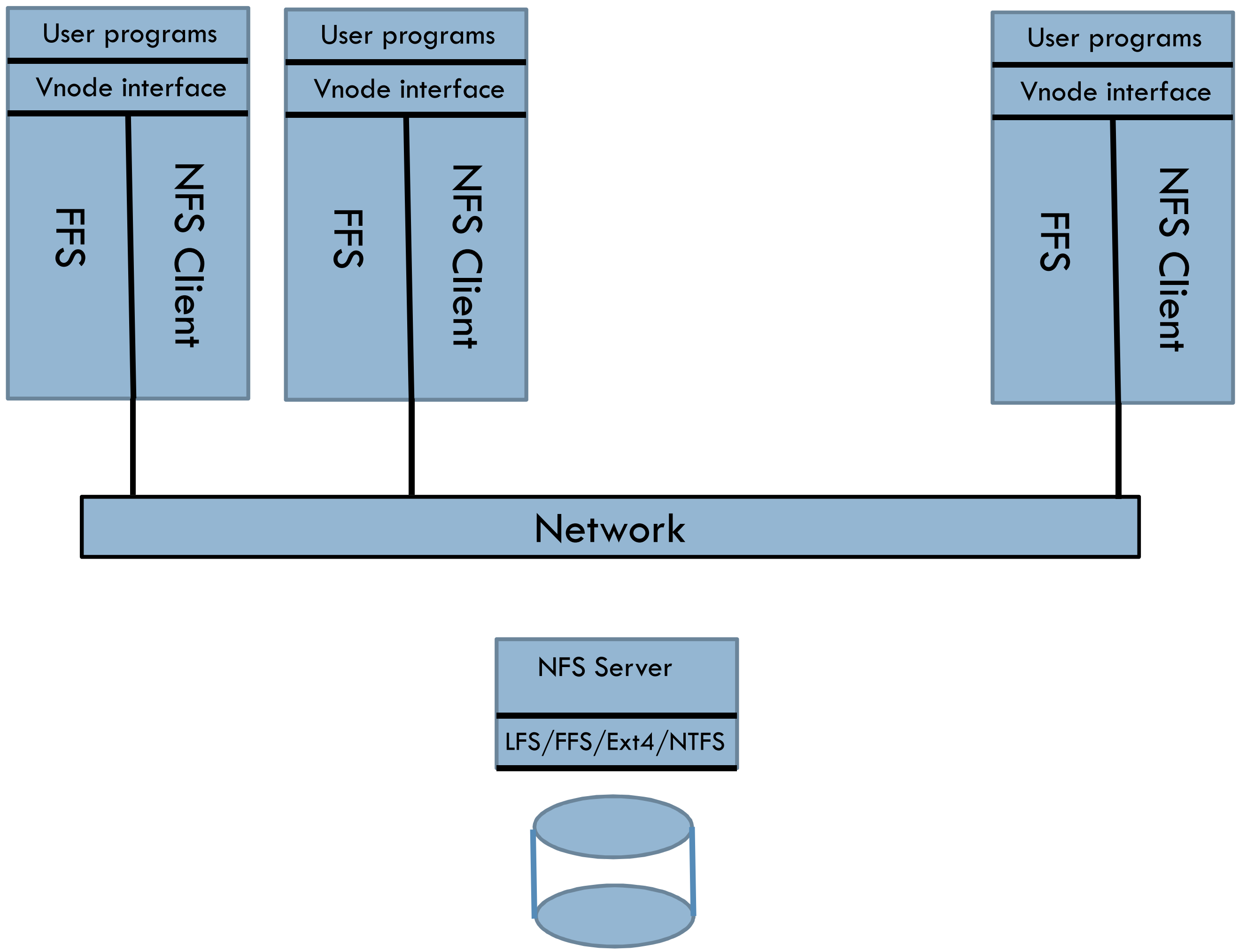
- Machine and OS independence
- Simple crash recovery for both clients and servers
- Transparent access to files
  - What does this mean?
- Provide UNIX-semantics to client
- “Reasonable” performance
  - What do they mean?

Overall design

- Motivate the VFS/vnode design
  - What was the vnode interface? Why have it?

Overall design

- Motivate the VFS/vnode design
  - Virtual File System (VFS): encapsulates operations on file systems (mount, unmount, sync)
  - Virtual Node (Vnode): encapsulates objects within a file system (read, write)
  - Advantage: separate generic FS operations from specific implementation


User programs
Vnode interface
FFS
NFS Client
User programs
Vnode interface
FFS
NFS Client
User programs
Vnode interface
FFS
NFS Client
Network
NFS Server
LFS/FFS/Ext4/NTFS

Virtual FS Module

- Allows processes to access file via descriptors
  - Just like local Unix files
  - For a given file access, decides whether to route to local FS or to NFS client system
- Names all files (local or remote) uniquely using "NFS file handles"
- Keeps a data structure for each mounted FS
- Keeps a data structure (the v-node) for all open files
  - If local file, v-node points to local disk i-node
  - If remote, v-node contains address of remote NFS server

NFS Protocol

- A small number of calls
  - Largely OS independent
  - Implementation used Sun RPC for communication layered on UDP
  - XDR for data representation
- Servers are stateless
  - What does this mean?
  - Advantages?
- File handles: used to identify files in messages (fsid, file id, generation number)
- Protocol routines: very similar to v-node ops

NFS RPC

- RPC: Remote Procedure Call
- Layered on UDP, XDR (marshalling layer)
  - Think Google protocol buffers
- RPCs were supposed to be idempotent
  - Servers didn't keep track of past requests
- Clients retransmitted till they got a reply back

NFS Server

- "Stateless"
  - Refers to connection state, not file data
  - Statelessness simplifies crash recovery
- Servers synchronously wrote data (and metadata) to their local file systems
  - Performance?
- Add generation numbers to distinguish newly created files from old files
- Server's local file system can be mounted by programs!
  - i-nodes could disappear under the NFS server!

NFS Client

- Typically built below the vnode layer of the kernel
  - Vnode = virtual inode, vnode refers to inode for local FS or to file handles for NFS
- Behaved like a local file system
  - Programs were unaware of the difference
  - Cached data, attributes, direntries
- Pathname traversals below the vnode layer
  - Client machine could mount several different file systems from different NFS servers

NFS Recovery

- What happens on a server crash?
- What happens on a client crash?

NFS Recovery

- Server crashes
  - No state kept on server
  - Recover local file system and the server is back online
  - Client will keep retrying
- Client crash
  - Loses cached data (if any)
  - No effect on server
- "Stupid server, smart client"

Implementation issues

- Convert kernel to vnodes
  - Identify all places that use inodes explicitly
  - Convert all calls to jump through vnodes
  - Rewrite namei to use vnode op (lookup)
  - Abstraction cost up to 2% in performance
- Add RPC and XDR to system
  - Took about 3 months
  - Tuned RPC round-trip to 8.8 ms
- Write the XDR routines that implement the NFS protocol
  - Modify kernel to do synchronous writes
  - Build mount protocol; break out from NFS
  - Two types of mount: hard and soft (retry or fail)
  - Implement user-level nfsd daemons (nfsd)

Challenging issues

- Root file systems; no NFS-mounted root
- Authentication based on uid/gid
  - Assumes consistent mappings across machines
  - Provoked development of yp
- Turn off root mapping on most machines
- No network locking (still no good solution)
- Deletes while file open: implemented as rename, delete on close (leaves garbage around in case of crashes)
- Time skew can be problematic

Performance

- Base performance on common UNIX utilities (compile, tbl, nroff, f77, sort, matrix inversion, make)
- Measurements: number of runs? Standard deviations?
- Improvements (basic engineering):
  - Client caching
  - Enlarge UDP packets (2K to 9000)
  - Remove one bcopy from path length
  - Added client attribute cache
  - Read-ahead small executables
  - Added name caching
  - Multiple getattr hack

Some Corners NFS Cut

- Security model
  - Cient OSs trusted
  - Client can impersonate others
- No coherent caching
  - Two clients could see different copies of the same file
- File locking not implemented initially
  - Later, lockd for advisory locks
- Did not support exact Unix file open semantics

NFS Caching Model

- Multiple clients could cache a file/directory for read/write
- Open/close caching
  - On close, client flushes all data to server
  - On open, client check attributes for change
    - Attributes refreshed periodically
  - Supports most applications adequately
- A single client's updates might change attributes
  - Client has no way of telling

Current NFS

- Allows root file system mounts
- Server write-behind
- Added stateful protocol
- Better crash recovery
- Can layer on UDP or TCP
- Added strong security

Write-behind in NFS

- Server can return without synchronously writing data
  - Returns “write verifier” token
- Client can force a write
  - Server returns “write verifier” token
- Client must buffer writes, until it knows server has written
- On server crash, server loses data, but client has it
  - Client forces a write
  - Users “write verifier” sequence number that the server changes on each crash

Scale and Peformance in a Distributed File System (1988)

- What kind of paper is this?
  - New big idea?
  - Measurement paper?
  - Experiences/lessons learnt paper?
  - A system description?
  - Performance study?
  - Refute-conventional wisdom?
  - Survey paper?

Scale and Peformance in a Distributed File System (1988)

- What kind of paper is this?
  - Retrospective

Overview

- Give system overview
- Define a benchmark to measure distributed performance
- Measure VICE-I
- Summarize problems in VICE-1
- Discuss VICE-II
- Measure VICE-II

The Andrew Benchmark

- What does it measure/compare?

The Andrew Benchmark

- Goal: compare local and remote execution times to understand the impact of scale and distribution
- Dataset size: 70 files; 200 KB
- Five phases
  - 1. Make Dir: construct a target subtree
  - 2. Copy: copy each file into target subtree
  - 3. ScanDir: traverse hierarchy, obtaining stat info
  - 4. ReadAll: read every byte
  - 5. Make: Compile and link the application

The Andrew Benchmark

- Results of benchmark
  - Shared tree 70% slower than local tree
  - TestAuth saturated at about 5 load units
  - CPU utilization was peaking above 75% on servers
- Conclusion: overall architecture is OK, but implementation could use some work
- Use benchmark results to moticate VICE-I to VICE-II redesign

Major Changes

- Cache Management: callbacks
  - What are these?
- Naming: FIDs
  - How does this help?
- Server process structure
  - Multi-threaded process instead of per-client process
- Low-level file system
  - Built new access-by-inode syscalls into UNIX

Consistency model

- Writes are visible immediately locally; remotely in a delayed fashion
- Upon close, writes are visible everywhere (except to existing opens)
- All other operations are globally visible
  - E.g., protection changes
- Workstations can operate on a file concurrently; no locking is provided

New performance numbers

- Changed clients!
- Shared files only 20% slower than local
- Scale to 20 clients with slowdown of 2X
- Callbacks eliminate most server interaction on ScanDir and ReadAll
- Scalability results are impressive: 70% CPU utilization at 20 load units

Comparison with NFS

- NSF is remote-open system
  - i.e., not whole-file caching like AFS
- Run the Andrew benchmark on both systems
- NFS time-outs improperly handled by apps, result in errors
- Paper results show AFS is superior to NFS except at very low load
- Andrew claims superior scalability

Operability

- Volumes: small grouping of files
- Map volumes to users
- Multiple volumes to a disk partition
- Can move volume just by updating volume DB
- Move volumes by creating clones, moving clone, repeating until there are no more updates
- Quotas enforced per volume
- Backups handled via clones

AFS evolved into Coda

- With proliferation of laptops in mid 1990s
  - AFS users often went a long time without any communication between desktop client and any AFS
  - Why not use AFS-like implementation when disconnected from the network
    - On a plane, at home, during network failure
  - Issues:
    - Which files to get before disconnection
    - consistency

Hoarding in Coda

- AFS keeps recently used files on local disk
  - Most of what you need will be around
- Users can specify "hoard lists" to tell Coda to cache a bunch of other things even if not already stored locally
- System can also learn over time which files a user tends to use

Consistency

- What if two disconnected users write the same file at the same time?
  - No way to use callback promises since server and client cannot communicate
- Coda's solution: cross your fingers, hope it does not happen and pick up pieces if it does
  - Log of changes kept while disconnected
  - Apply changes upon reconnect
  - If conflict detected, try to resolve automatically, else ask the user
- In practice, unfixable conflicts almost never happen