Networked File System

CS333 S20 :: Williams College

Course Logistics

Lab 3a

- Teams, repos, panics, due date

24-hour take-home midterm

- Will be in envelopes in my mailbox

Last Class

LFS

- No-overwrite file system
- LFSes work hard to optimize writes by avoiding seeks
  - Rely on caching to avoid I/O costs for fragmented data during reads
- Has some not-so-unlikely worst cases
  - Sequential-read-after-random-write
  - Full-disk behavior? High segment cleaning overheads…
- Garbage collection
  - Hard to evaluate the cost of garbage collection because workload dependent
  - Introduces ***I/O amplification***: system I/O greatly exceeds user I/O

This Class

NFS: Network File System

- Discussion
- Handout

FUSE FS

- Define data structures
- Discuss utility methods
- structs, unions, void *, typedefs, etc.

If extra time:

- Activity (NFS trace analysis)

Evolving Protocol

NFS has evolved over many years:

- NFS v1: internal SUN protocol
- NFS v2: 1989
- NFS v3: 1995
- NFS v4: 2000
- NFS v4.1: 2010
- NFS v4.2: 2016

What's changed?

- Statelessness has given way to statefulness in NFSv4
  - The "purity" of the v2 design has eroded in favor of performance & security

Concepts/Questions

Statelessness vs. statefulness?

What is idempotency and why are idempotent operations desirable?

Who caches what, and what are the implications?

What consistency guarantees does NFS give to clients?

What happens when an NFS client crashes? An NFS server?

NFS v2: High-level idea

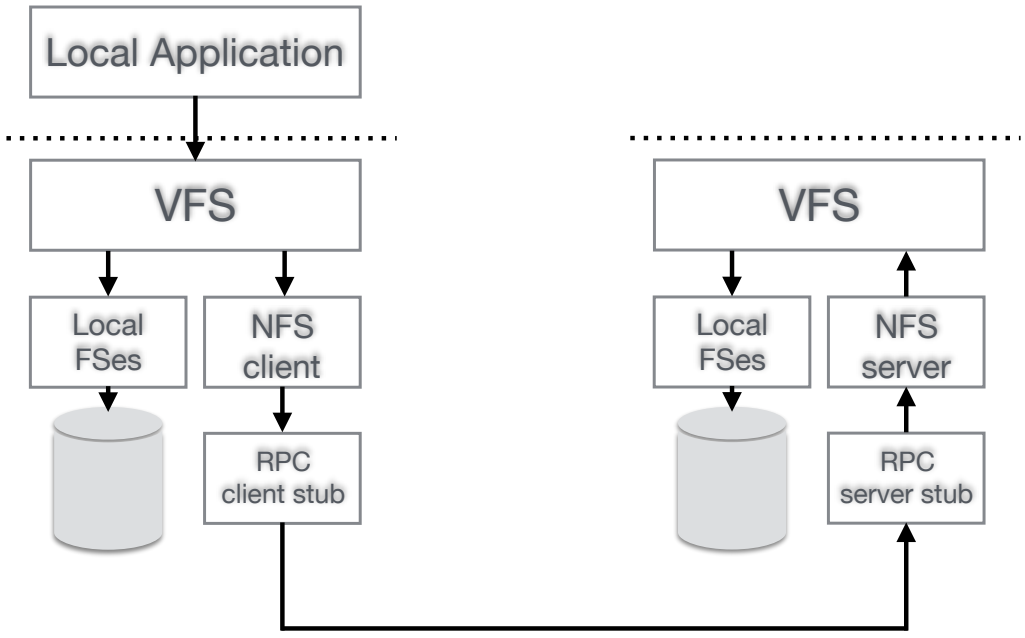
Clients provide all state with their requests

- File handle: volume number, inode number, generation number

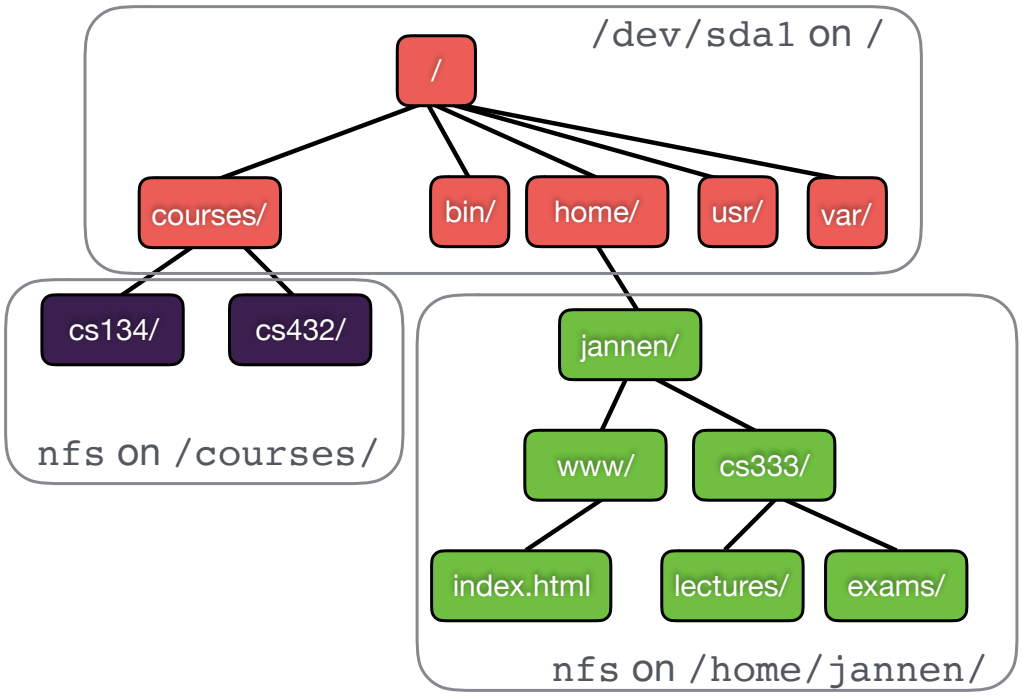
Local VFS operations translated into a series of network requests

The fact that NFS uses the client-server model is transparent to applications

NFS Big Picture

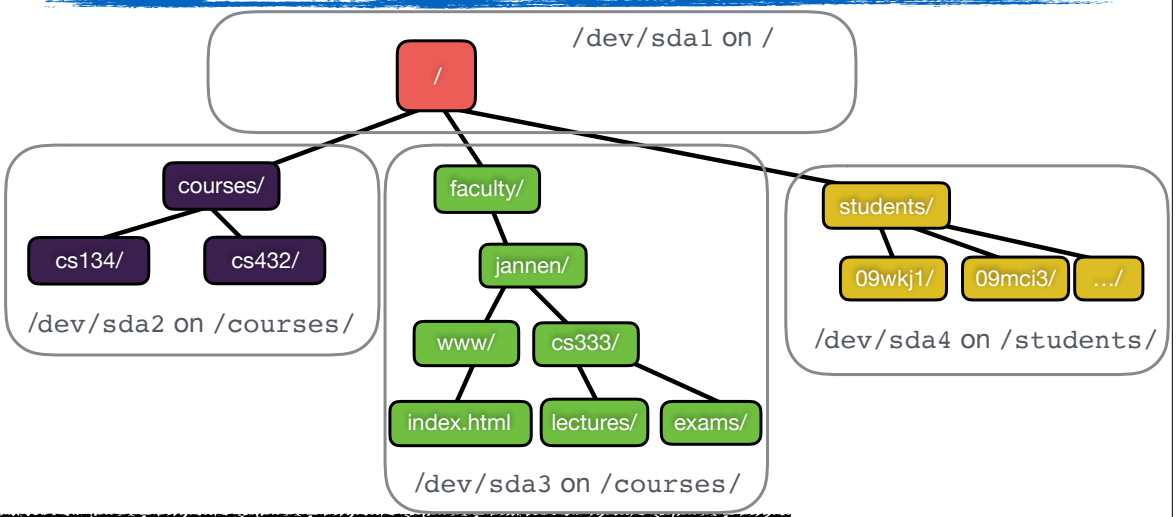


NFS Big Picture: Client



Clients mount nfs volumes into their FS namespace

NFS Big Picture: Server



NFS servers export subtrees to clients.

/etc/exports contains nfs server settings:

- `/faculty/jannen` exported RW to userid:jannen on dexter.cs.williams.edu, speckle.cs.williams.edu, …
- `/courses/`exported RW to userid:jannen on dexter.cs.williams.edu, speckle.cs.williams.edu, …
- `/students/09mci3/`exported RW to userid:09mci3 on dexter.cs.williams.edu, speckle.cs.williams.edu, …

NFSv2 Statelessness

POSIX is stateful, NFSv2 is not

- What challenges does this bring?

An NFSv2 server has no notion of a "connection"

- The client bundles all necessary "state" into each message
  - NFS **file handle** stores volume, inode #, generation #
- The server can then view each client message in isolation
  - If a client disconnects, the server does not know or care
  - If the server crashes, the client does not lose any relied-upon state

NFSv2 Statelessness

Idempotent operations

- An operation is idempotent if performing the operation multiple times has the same effect as performing it once
  - Many POSIX system calls are not idempotent:
    - read(fd, buf, nbytes), write(fd, buf, nbytes), lseek(fd, offset, whence), etc.
  - By specifying the starting offset with each read/write, many operations can be made idempotent
    - Notable exceptions?

Idempotency simplifies the protocol considerably

- If a message is dropped, send it again
- If the server crashes, clients can resend all unacknowledged operations

Caching: Who and What?

Network round trips are expensive, so clients would like to cache data locally

- Satisfy reads from cache, rather than pay for read+RTT
- Buffer writes locally, and send updates in large messages

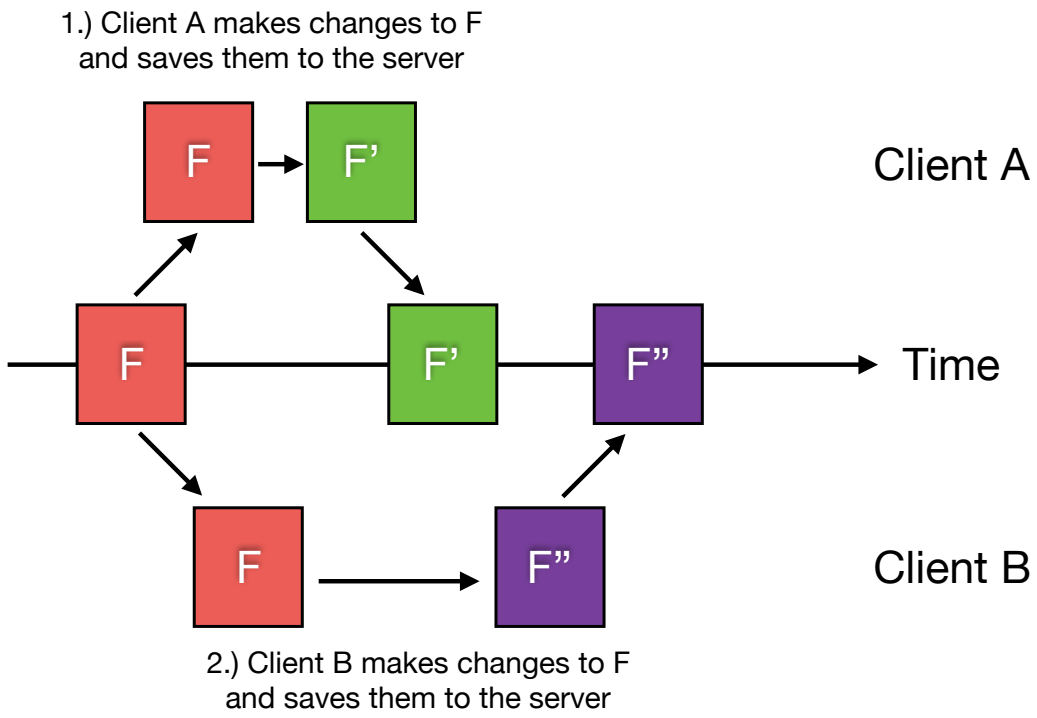
Multiple clients can't share data effectively without coordination

- NFSv2 is supposed to appear as a local FS
- When is local cache stale w.r.t. server?
- When should you push your updates to server?

NFSv2 solutions:

- Flush-on-close / close-to-open consistency
- Attribute cache

Close-to-open Consistency



Tying to Previous Topics

OSTEP 1.00 section 49.11, final paragraph:

> The problem that this requirement gives rise to in NFS server implementation is that write performance, without great care, can be the major performance bottleneck. Indeed, some companies (e.g., Network Appliance) came into existence with the simple objective of building an NFS server that can perform writes quickly; one trick they use is to first put writes in a battery-backed memory, thus enabling to quickly reply to WRITE requests without fear of losing the data and without the cost of having to write to disk right away; the second trick is to use a file system design specifically designed to write to disk quickly when one finally needs to do so [HLM94, RO91].

- NetApp Filers used WAFL: a log-structured "file system"!

Activity: NFS Trace Analysis

https://github.com/williams-cs/cs333-class

Activity: FUSE FS Design

What are the important structures?

What are the important utilities?

What is a good implementation plan?