UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537
Introduction to Operating Systems

Andrea C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

Distributed File Systems

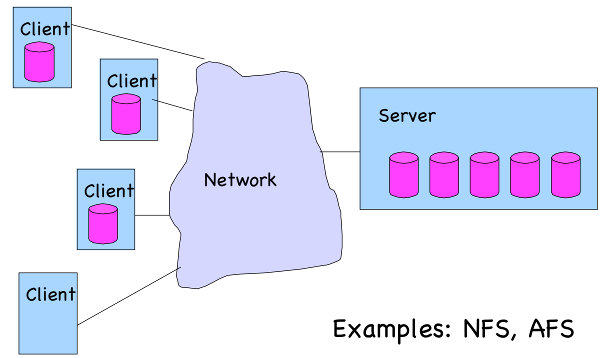
Questions answered in this lecture:

Why are distributed file systems useful?

What is difficult about distributed file systems?

How does NFS work?

What is a distributed file system?



Examples: NFS, AFS

Motivation

Why are distributed file systems useful?

- Access from multiple clients
  - Same user on different machines can access same files
- Simplifies sharing
  - Different users on different machines can read/write to same files
- Simplifies administration
  - One shared server to maintain (and backup)
- Improve reliability
  - Add RAID storage to server

Challenges

Transparent access
- User sees single, global file system regardless of location

Scalable performance
- Performance does not degrade as more clients are added

Fault Tolerance
- Client and server identify and respond appropriately when other crashes

Consistency
- See same directory and file contents on different clients at same time

Security
- Secure communication and user authentication

Tension across these goals
- Example: Caching helps performance, but hurts consistency

Case Study: NFS

Sun's Network File System

- Introduced in 1980s, multiple versions (v2, v3, v4)

Key idea #1: Stateless server

- Server not required to remember anything (in memory)
  - Which clients are connected, which files are open, ...
- Implication: All client requests have enough info to complete op
  - Example: Client specifies offset in file to write to
- One Advantage: Server state does not grow with more clients

Key idea #2: Idempotent server operations

- Operation can be repeated with same result (no side effects)
- Example: a=b+1; Not a=a+1;

Helps which Challenge??

NFS Overview

Remote Procedure Calls (RPC) for communication between client and server

Client Implementation

- Provides transparent access to NFS file system
  - UNIX contains Virtual File system layer (VFS)
  - Vnode: interface for procedures on an individual file
- Translates vnode operations to NFS RPCs

Server Implementation

- Stateless: Must not have anything only in memory
- Implication: All modified data written to stable storage before return control to client
  - Servers often add NVRAM to improve performance

Basic NFS Protocol

Operations

- `lookup(dirfh, name) returns (fh, attributes)`
  - `Use mount protocol for root directory`
- `create(dirfh, name, attr) returns (newfs, attr)`
- `remove(dirfs, name) returns (status)`
- `read(fh, offset, count) returns (attr, data)`
- `write(fh, offset, count, data) returns attr`
- `gettattr(fh) returns attr`

What is missing??

Mapping UNIX System Calls to NFS Operations

Unix system call: `fd = open("/dir/foo")`

- Traverse pathname to get filehandle for foo
  - `dirfh = lookup(rootdirfh, "dir");`
  - `fh = lookup(dirfh, "foo");`
- Record mapping from `fd` file descriptor to `fh` NFS filehandle
- Set initial file offset to 0 for `fd`
- Return `fd` file descriptor

Unix system call: `read(fd,buffer,bytes)`

- Get current file `offset` for `fd`
- Map `fd` to `fh` NFS filehandle
- Call `data = read(fh, offset, bytes)` and copy `data` into `buffer`
- Increment file `offset` by `bytes`

Unix system call: `close(fd)`

- Free resources assocatiated with fd

Client-side Caching

Caching needed to improve performance

- Reads: Check local cache before going to server
- Writes: Only periodically write-back data to server
- Avoid contacting server
  - Avoid slow communication over network
  - Server becomes scalability bottleneck with more clients

Two client caches

- data blocks
- attributes (metadata)

Cache Consistency

Problem: Consistency across multiple copies (server and multiple clients)

- How to keep data consistent between client and server?
  - If file is changed on server, will client see update?
  - Determining factor: Read policy on clients
- How to keep data consistent across clients?
  - If write file on client A and read on client B, will B see update?
  - Determining factor: Write and read policy on clients

NFS Consistency: Reads

Reads: How does client keep current with server state?

- Attribute cache: Used to determine when file changes
  - File open: Client checks server to see if attributes have changed
    - If haven't checked in past T seconds (configurable, T=3)
  - Discard entries every N seconds (configurable, N=60)
- Data cache
  - Discard all blocks of file if attributes show file has been modified

Eg: Client cache has file A's attributes and blocks 1, 2, 3

- Client opens A:
- Client reads block 1
- Client waits 70 seconds
- Client reads block 2
- Block 3 is changed on server
- Client reads block 3
- Client reads block 4
- Client waits 70 seconds
- Client reads block 1

NFS Consistency: Writes

Writes: How does client update server?

- Files
  - Write-back from client cache to server every 30 seconds
  - Also, Flush on close()
- Directories
  - Synchronously write to server

Example: Client X and Y have file A (blocks 1,2,3) cached

- Clients X and Y open file A
- Client X writes to blocks 1 and 2
- Client Y reads block 1
- 30 seconds later...
- Client Y reads block 2
- 40 seconds later...
- Client Y reads block 1

Conclusions

Distributed file systems

- Important for data sharing
- Challenges: Fault tolerance, scalable performance, and consistency

NFS: Popular distributed file system

- Key features:
  - Stateless server, idempotent operations: Simplifies fault tolerance
  - Crashed server appears as slow server to clients
- Client caches needed for scalable performance
  - Rules for invalidating cache entries and flushing data to server are not straight-forward
  - Data consistency very hard to reason about