Distributed Filesystems

CS 475, Spring 2018
Concurrent & Distributed Systems
HW2 Discussion

HW2 Submissions per day, as of Sun Feb 25 13:35:13 2018. Total = 3,058
HW2 Discussion

HW2 Submissions per-student, as of Sun Feb 25 13:36:31 2018, mean = 58
Review: Domain Name System

Global Layer
- net
- org
- edu
- com
- gov
- uk

Administrational Layer
- root-servers
- gmu

Managerial Layer
- www
- cs
- www
Review: Domain Name System - Scale

980 physical servers

Global Layer

Administrational Layer

Managerial Layer

net (13)
org (6)
edu (6)
com (13)
gov (4)
uk (8)

root-servers (3)
gmu (4)

www (1)
www (1**)

cs (3*)

www (1**)
Review: Multicast

- Multicast increases the efficiency of networks
  - Point-to-point broadcast requires the sender to send $N$ copies of the message
  - Multicast broadcast only sends one copy
  - Network switches replicate the traffic faster and more efficiently
  - Unicast: 15 copies
Announcements

• HW3 is out!
• Today: Distributed Filesystems
  • Abstraction
  • What leaks through
  • Implementation tradeoffs
• Additional reading: OS TEP Ch 49
Files

• File:
  • Name
  • Size (bytes)
  • Create/Access/Modification Time
  • Contents (binary)

• Directory:
  • Maintains a list of the files (and their metadata) in that directory
File Operations

- Create
- Write – at write pointer location
- Read – at read pointer location
- Reposition within file - seek
- Delete
- Truncate
- Open($F_i$) – search the directory structure on disk for entry $F_i$, and move the content of entry to memory
- Close ($F_i$) – move the content of entry $F_i$ in memory to directory structure on disk
Directory Operations

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system
Open file locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - Shared lock similar to reader lock – several processes can acquire concurrently
  - Exclusive lock similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - Mandatory – access is denied depending on locks held and requested
  - Advisory – processes can find status of locks and decide what to do
Directory Structure

• Directories contain information about the files in them
• Directories can be nested
• Operations on directories:
  • Create file
  • List files
  • Delete file
  • Rename file
Filesystems

- Define how files and directory structure is maintained
- Exposes this information to the OS via a standard interface (driver)
- OS can provide user with access to that filesystem when it is **mounted**
  - (Example: NFS, AFP, SMB)
Filesystem Functionality

• Directory management (maps entries in a hierarchy of names to files-on-disk)
• File management (manages adding, reading, changing, appending, deleting) individual files
• Space management: where on disk to store these things?
• Metadata management
Mounting Filesystems

Filesystem driver is passed path *only* from its mount point (e.g. it doesn’t care where it is mounted)
Distributed File Systems

- Goals
  - Shared filesystem that will look the same as a local filesystem
  - Scale to many TB’s of data/many users
  - Fault tolerance
  - Performance
Distributed File Systems

• Challenges:
  • Heterogeneity (different kinds of computers with different kinds of network links)
  • Scale (maybe lots of users)
  • Security (access control)
  • Failures
  • Concurrency
Strawman Approach

• Use RPC to forward every filesystem operation to the server
• Server serializes all accesses, performs them, and sends back result.
Strawman Approach

Client

open(“file”)

fd

seek(fd, 10)

read(fd)

Server
Strawman Approach

• Use RPC to forward every filesystem operation to the server
• Server serializes all accesses, performs them, and sends back result.
• Great: Same behavior as if both programs were running on the same local filesystem!
• Bad: Performance can stink. Latency of access to remote server often much higher than to local memory
NFS

- Cache file blocks, file headers, etc., at both clients and servers.
- Advantage: No network traffic if open/read/write/close can be done locally.
- But: failures and cache consistency.
- NFS trades some consistency for increased performance... let’s look at the protocol.
Problem: read() depends on server remembering that client did seek()!

How to solve?
NFS + Server crash?

- Data in memory but not disk lost
- So... what if client does `seek() ; /* SERVER CRASH */; read()`
  - If server maintains file position, this will fail. Ditto for `open()`, `read()`
- Stateless protocol: requests specify exact state. `read() -> read( [position])`. no seek on server.
NFS + Server Crash

Client

open(“file”)

fd

seek(fd, 10)

read(fd, offset)

read is correct because server doesn’t keep track of any state

Server

CRASH!

Reboot!
NFS + Lost Messages?

• Lost messages: what if we lose acknowledgement for delete(“foo”)

• And in the meantime, another client created foo a new file called foo?

• Solution: Operations are idempotent
  • How can we ensure this? Unique IDs on files/directories. It’s not delete(“foo”), it’s delete(1337f00f), where that ID won’t be reused.
NFS + Client Crashes

• Might lose data in client cache
• Doesn’t matter:
  • If lose other people’s data, can always retrieve it again
• Local writes go to cache until close() is called and returns (which flushes to server)
• If lose your own writes sooner, SOL
NFS Failure Handling

• You can choose -
  • retry until things get through to the server
  • return failure to client

• Most client apps can’t handle failure of close() call. NFS tries to be a transparent distributed filesystem -- so how can a write to local disk fail? And what do we do, anyway?

• Usual option: hang for a long time trying to contact server
NFS Failure Handling

• Not everything is idempotent! Some stuff leaks through!

Client

Server

mkdir(“dir”)

mkdir(“dir”)

OK

error: “dir” exists
Cache Consistency: Update Visibility

1. Read File: “a”
2. Write File: “b”

Update Visibility: When do Client 2’s writes become apparent to the server?
Cache Consistency: Stale reads

1. Read File: “a”

2. Write File: “b”

Stale reads: Once the server gets updated, how does client 1 know that File 1 has been updated?
Cache Consistency Strawman

- Before any read(), ask server if file has changed
  - If not, use cached version
  - If so, get fresh data from server
- Bad news: floods the server with requests
- Anyway: this alone is not enough to make sure each read() sees the latest write()
  - How do we know when the write() gets committed?
NFS Caching - Close-to-open

• Implemented by most NFS clients
• Contract:
  • if client A write()s a file, then close()s it,
  • then client B open()s the file, and read()s it,
  • client B’s reads will reflect client A’s writes
• Benefit: clients need only contact server during open() and close()—not on every read() and write()
NFS Caching - Close-to-open

1. Open File
2. Read File: “a”

Client 1 cache

3. Open File
4. Write File: “b”

Client 2 cache

File 1: “b”

Server

File 1: “b”

5. Open File
6. Read File: “a”

Client 4 cache

7. Close File

Client 3 cache

8. Open File
9. Read File: “b”

Note: in practice, client caches periodically check server to see if still valid
NFS + Locking

• Does NFS support locks?
• Nope! How could it support locks and still be stateless?
• Fault-tolerant lock servers are really hard to implement
• We’ll discuss in lectures 15-18
Sidebar: Heartbeat Protocols

• Allow client/server to remain aware of each other’s status
• For HW3: does client still have locks (client checking server, server checking client)
• For NFS: is cache still valid? (client checking server)

Diagram:

Client

lock("foo")

OK, stamp = 1

Hi, I’m stamp 1, still have foo

Hmm, I guess server is gone, maybe lock is not valid

Server

CRASH!
Sidebar: Heartbeat Protocols

- Allow client/server to remain aware of each other’s status
- For HW3: does client still have locks (client checking server, server checking client)
- For NFS: is cache still valid? (client checking server)
Sidebar: Heartbeat Protocols

- We call these time-limited locks **leases**
- What does a lease guarantee?
  - If no network failures
    - Locks that are relinquished when client crashes
  - If network failures/delays:
    - Nothing
NFS Security

• What prevents unauthorized users from issuing RPCs to an NFS server?
• What prevents unauthorized users from forging NFS replies to an NFS client?
  • **Nothing: IP-address based security only. Client A can access mount M. That’s it!**
NFS Limitations

• Security: what if untrusted users can be root on client machines?
• Scalability: how many clients can share one server?
  • Writes always go through to server
  • Some writes are to “private,” unshared files that are deleted soon after creation
• Can you run NFS on a large, complex network?
  • Effects of latency? Packet loss? Bottlenecks?
Other Approaches

• What about handling hundreds of thousands of concurrent clients and exabytes of data?

• We will discuss GFS, the Google File System in lecture 20 - it does exactly this!