EECS 491
Introduction to Distributed Systems

Fall 2019

Harsha V. Madhyastha
RSM with Paxos

- Replicated log with separate Paxos instance per slot
  - Writes: Append to log
  - Reads: Catch up and then execute read

- Performance enhancements:
  - Leader-based Paxos to skip Prepare phase
  - Read leases to enable local reads
RSM with Primary Backup

State at each layer

Balance for each account

Chosen op for each slot

Want to replace replica R with new replica R’

What could go wrong if no state transferred from R to R’?

What state must we transfer?

RSM

Application

Paxos

n_p, n_a, v_a
Bootstrapping new replica

- Unsafe to start new replica with empty state
  - Chosen ops may no longer have majority acceptance

- Redundant to transfer app *and* Paxos state
  - App state constructed by applying ops in log

- Transferring *only* app state insufficient
  - Replica may not have learnt all chosen operations

- Enough to transfer Paxos state, but can be huge
Garbage collection

- When can we garbage collect Paxos state?
- Throw away state for a slot once all replicas confirm they know chosen op for that slot
- Server’s state = Paxos log + App state after applying garbage collected portion of log
- Need another round of communication after Learn phase
- Like Learn phase, this round is asynchronous
Changing set of replicas

- Initial set of replicas: R1, R2, R3
- Want to replace R3 with R4

- Send config message to all four:
  - “New replica set is R1, R2, R4”

- What could go wrong?
- R1 and R4 get config message
- R1 and R4 accept op1 for a slot
- R2 and R3 accept op2 for same slot!
Changing set of replicas

- Initial set of replicas: R1, R2, R3
- Want to replace R3 with R4

- \( C_{\text{old}} = \) old replica set, \( C_{\text{new}} = \) new replica set
- Get majority of \( (C_{\text{old}} \cup C_{\text{new}}) \) to agree to \( (C_{\text{old}} \cup C_{\text{new}}) \) as replica set
- Get majority of \( C_{\text{new}} \) to agree to \( C_{\text{new}} \) as replica set
- No point in time when \( C_{\text{old}} \) and \( C_{\text{new}} \) can come to consensus independently on same slot
Project 2

- Write macro test cases
- Mimic all complexities
  - Many clients issuing PutAppends in parallel
  - Unreliable network
  - Servers restart
  - Servers fail permanently

- How to verify correctness?
  - Confirm no violation of linearizability
Case Studies using Paxos

- Chubby
- ZooKeeper
Building Chubby was an engineering effort required to fill the needs mentioned above; it was not research. We claim no new algorithms or techniques. The purpose of this paper is to describe what we did and why, rather than to advocate it. In the sections that follow, we de-
Chubby

- Internal service within Google
- Highly available coordination service

Two purposes:
- Lock service
- File system (for small files)
x = Open("/ls/gfs-cell8/chunkmaster")

if (TryAcquire(x) == success) {
  // I'm the chunkmaster, tell everyone
  SetContents(x, my-address)
}

else {
  // I'm not the master, find out who is
  chunkmaster = GetContents(x)
}
Chubby Design

Replicated service using Paxos to implement fault-tolerant log

Diagram showing client application and library interacting with 5 servers of a Chubby cell through RPCs.
Why this interface?

- Why not a library that implements Paxos?
- Developers do not know how to use Paxos
  - They at least think they know how to use locks!
- Need to export result of coordination outside of system
  - Example: Clients need to discover GFS master
Reads/Writes in Chubby

- One of the 5 replicas chosen as the master
- Clients submit all reads and writes to master
- How to handle master failure?
  - Another replica must propose itself as master
  - New master must first “catch up”
- Master is performance bottleneck
Scaling Chubby

- Clients cache data they read
- Reading from local cache violates linearizability
  - How to fix this?

- Master invalidates cached copies upon update
  - While invalidation in progress, respond to reads with uncachable response

- Master must store knowledge of client caches
- What if master fails?
Scaling Chubby with Proxies

Proxy

Proxy

Chubby
Handling Client Failures

- What if a client acquires a lock and fails?
- Client library exchanges keep-alives with master
- Lock revoked upon client failure
- Problem?
  - Network partition not client failure
- Chubby associates lock acquisitions with sequence numbers
  - Can distinguish operations from previous lock holders
Discrepancy in Lock Validity

Check with lock service to confirm lock validity
Handling Master Failure

- With every keep-alive message, master grants lease to client
  - Client lease valid at most until master lease valid

- When client lease expires, client disables cache
  - Keep-alive with new master might re-enable cache

- Thus, okay to lose client cache info at master
ZooKeeper

- Open source coordination service
- Addresses need for polling in Chubby
  - Example: *If you cannot acquire lock, need to retry*
- Goal in ZooKeeper: *Wait-free coordination*
ZooKeeper: Watch mechanism

- Clients can register to “watch” a file
  - ZooKeeper notifies client when file is updated

- Example:
  - Try to acquire a lock by creating a file
  - If file already exists, watch for updates
  - Upon watch notification, try to re-acquire lock

- Problem?
  - Herd effect
ZooKeeper: Hierarchy

/app1

/app1/p_1  /app1/p_2  /app1/p_3

/app2
ZooKeeper: Sequencer

**Lock**
1. \( n = \text{create}(1 + \text{"/lock-"}, \text{EPHEMERAL|SEQUENTIAL}) \)
2. \( C = \text{getChildren}(l, \text{false}) \)
3. \textbf{if} \( n \) is lowest znode in \( C \), exit
4. \( p = \text{znode in } C \) ordered just before \( n \)
5. \textbf{if} \( \text{exists}(p, \text{true}) \) wait for watch event
6. goto 2

**Unlock**
1. delete(\( n \))
Chubby vs. ZooKeeper

- Difference between invalidation and watch?

- Invalidation:
  - Only library receives notification to update cache

- Watch:
  - Application receives notification
  - Only app knows what it needs to do