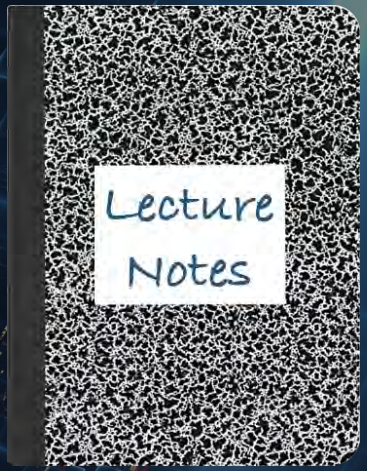
CS 417 – DISTRIBUTED SYSTEMS

Week 6: Chubby



Paul Krzyzanowski

Some issues in distributed systems design

1. You might need a coordinator
   - But implementing leader election algorithms could be a pain
2. You might need to implement locking
   - Implementing distributed mutual exclusion is a pain
3. Your collection of programs may need to read configuration info
   - The information may change dynamically (e.g., a group membership list)

Running a central service for locking and object storing is super convenient

- But is not fault-tolerant
- Implementing the service as a replicated state machine will fix that but is a pain
  - Elections, consensus algorithm, RPCs, request redirection, recovery

Google Chubby

Designed as a distributed lock service + simple fault-tolerant file system

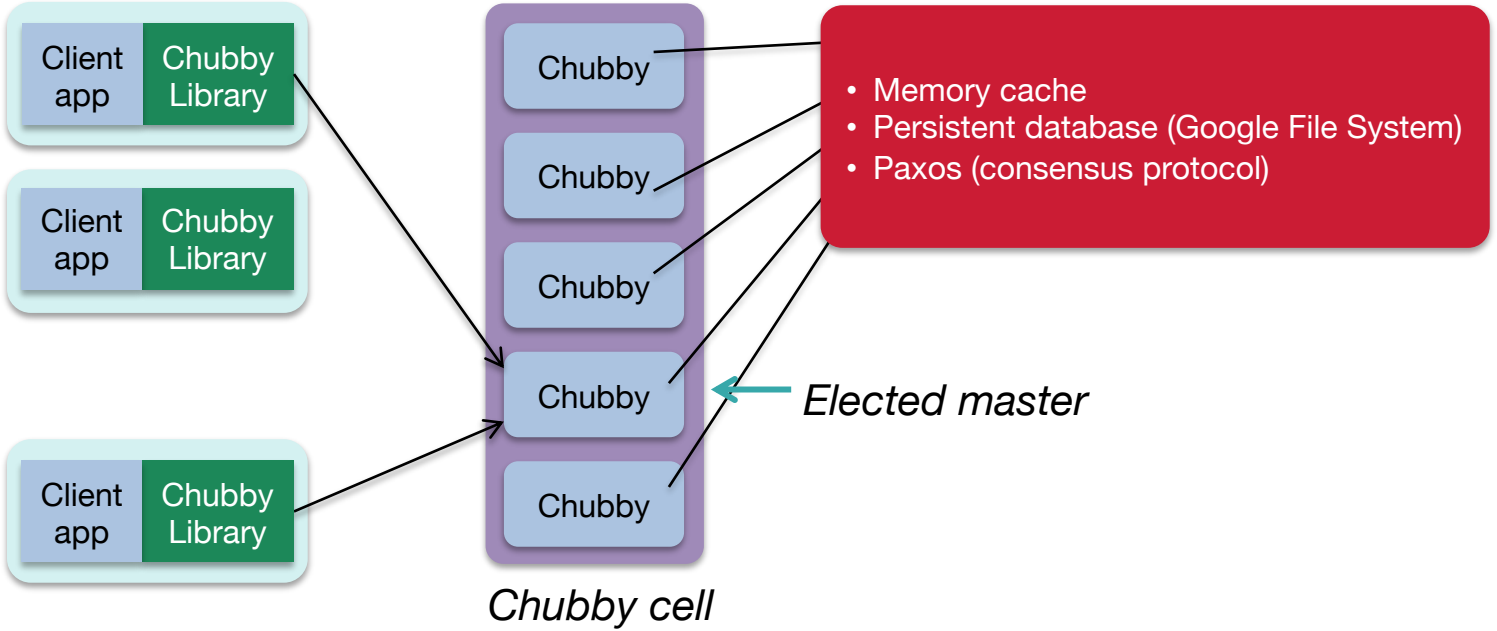
- Interfaces
  - File access
  - Event notification
  - File locking
- Chubby is used to:
  - Manage coarse-grained, long-term locks (hours or days, not < sec)
    - get/release/check lock – identified with a name
  - Store small amounts of data associated with a name
    - E.g., system configuration info, identification of primary coordinators
  - Elect masters

Design priority: availability rather than performance

Chubby Deployment

Client library + a Chubby cell (5 replica servers)

Primary/backup replication



Chubby Master

- Chubby has **at most** one master
  - All requests from the client go to the master

- All other nodes (replicas) must agree on who the master is
  - Paxos consensus protocol used to elect a master
  - Master gets a lease time
    - Re-run master selection after lease time expires to extend the lease

      …or if the master fails

Simple User-level API for Chubby

- User-level RPC interface
  - Not implemented as an OS-level file system
  - Programs must access Chubby via an API

- Look up Chubby nodes via DNS
- Ask any Chubby node for the master node
- File system interface (names, content, and locks)

Chubby: File System Interface

- /ls/cell/rest/of/name
  - /ls: lock service (common to all Chubby names)
  - cell: resolved to a set of servers in a Chubby cell via DNS lookup
  - /rest/of/name: interpreted within the cell

- Each file has
  - Name
  - Data
  - Access control list
  - Lock
  - No modification, access times
  - No seek or partial reads/writes; no symbolic links; no moves

Chubby: API

open()	Set mode: read, write & lock, change ACL, event list, lock-delay, create
close()	Close access to an object
GetContentsAndStat()	Read file contents & metadata
SetContents(), SetACL()	Write file contents or ACL
Delete()	Delete an object
Acquire(), TryAcquire(), Release()	Lock operations
GetSequencer()	Sequence # for a lock
SetSequencer()	Associate a sequencer with a file handle
CheckSequencer()	Check if sequencer is valid

Chubby: Files as Locks

- Every file & directory can act as a reader-writer lock
  - Either one client can hold an exclusive (writer) lock
  - Or multiple clients can hold reader locks
- Locks are advisory
- If a client releases a lock, the lock is immediately available
- If a client fails, the lock will be unavailable for a *lock-delay* period (typically 1 minute)

Using Locks for Leader Election

Using Chubby locks makes leader election easy

How: a participant tries to acquire a lock

- If it gets the lock, then it's the master for whatever service it's providing!

– No need for user servers to participate in a consensus protocol

… the programmer doesn't need to figure out consensus (e.g., Paxos or Raft)

– Chubby provides the fault tolerance

Example: electing a coordinator & using it to write to a file server

– Participant gets a lock, becomes coordinator

- Gets a lock sequence count from Chubby

– In each RPC to a server, send the sequence count to the server

– During request processing, a server will reject old (delayed) packets

```
if (sequence_count < current_sequence_count)
    reject request  // it must be from a delayed packet
```

Events

Clients may subscribe to events:

- File content modifications
- Child node added/removed/modified
- Chubby master failed over
- File handle & its lock became invalid
- Lock acquired
- Conflicting lock request from another client

Chubby client caching & master replication

- At the client
  - Data cached in memory by chubby clients
    - Cache is maintained by a Chubby lease, which can be revoked, invalidating the cache
  - All clients write through to the Chubby master
- At the master
  - Writes are propagated via Paxos consensus to all Chubby replicas
    - Data updated in total order – replicas remain synchronized
    - The master replies to a client *after* the writes reach a **majority** of replicas
    - Reads can be acknowledged immediately without consulting the replicas.
  - Cache invalidations
    - Master keeps a list of what each client may be caching
    - Invalidations are sent by the master and are acknowledged by the client
    - File is then cacheable again
  - Chubby database is backed up to GFS every few hours

Chubby – Apache ZooKeeper™

- Chubby is an internal Google service
- Apache ZooKeeper is an open-source centralized service for locking and storing shared services
- Built based on the Chubby paper
  - Uses Zab instead of Paxos for consensus
    (another protocol – roughly similar to Paxos but with a focus on log replication, like Raft)
  - Replicated servers
  - Shared hierarchical namespace
    - Stored in memory and organized into files and directories
  - Locking
  - Event notification

APACHE ZooKeeper™

Apache ZooKeeper™

- Different from Chubby – but inspired by Chubby & similar in many ways
  - Billed as a “process coordinator” rather than a “lock service”
- Adds “watch mechanism”
  - One-time event trigger to notify a client when a file is updated
- Chubby supports client-side caching with invalidation by the master
  - Zookeeper relies clients polling the master or using the watch mechanism to check for updates
  - Doesn’t guarantee strong consistency:
    - Client A changes the contents of a Chubby file and gets an acknowledgment
    - That doesn’t mean that all other clients have received notifications of the change even if they are watching the file → clients may have different ideas of the contents
  - This has been fixed by Netflix with *Curator*
    - JVM client library for ZooKeeper
    - Adds client caching, strong consistency, and simplified APIs for common tasks

The End