# The Chubby Lock Service for Loosely-coupled Distributed systems

Author: Mike Burrows
Presenter: Ke Nian

David R. Cheriton School of Computer Science,
University of Waterloo,

Waterloo, Canada

October 22, 2014

# Agenda

# Distributed Consensus Problem

- All participants in a distributed system or process must agree on the same value (state) despite of failures:
  - Node failures
  - Communication failures
- Examples:
  - Whether or now to commit a transaction to a Database
  - Who is master node of the system

# Distributed Consensus Problem
## Requirements





- Safeness: All nodes must agree on the same value or state
- Liveness: If less than some fraction of nodes crash, the rest of systems can still reach some agreements.

# Distributed Consensus Problem
## Solutions

- Two-Phase Commit(2PC):
  - Always safe
  - Can cause blocking
- Three-Phase Commit(3PC):
  - Not always safe
  - Will not cause blocking
- Paxos Protocol:
  - Always safe
  - Can cause blocking only in rare situation

# Overview of Chubby
## Chubby in a nutshell

Chubby reduce the distributed consensus problem to keeping a consistent namespace and its corresponding contents using a lock service.

# Overview of Chubby
## What Chubby Provides?



Chubby provides

- A ***coarse-grained advisory*** lock service.
- A reliable storage for a loosely-coupled distributed system.
- A client interface to provide notifications of various events.

# Overview of Chubby
## How is Chubby Used?

Chubby is used for:

- Synchronization of activities such as electing primary
- Acting as name servers
- Storing meta data for other systems

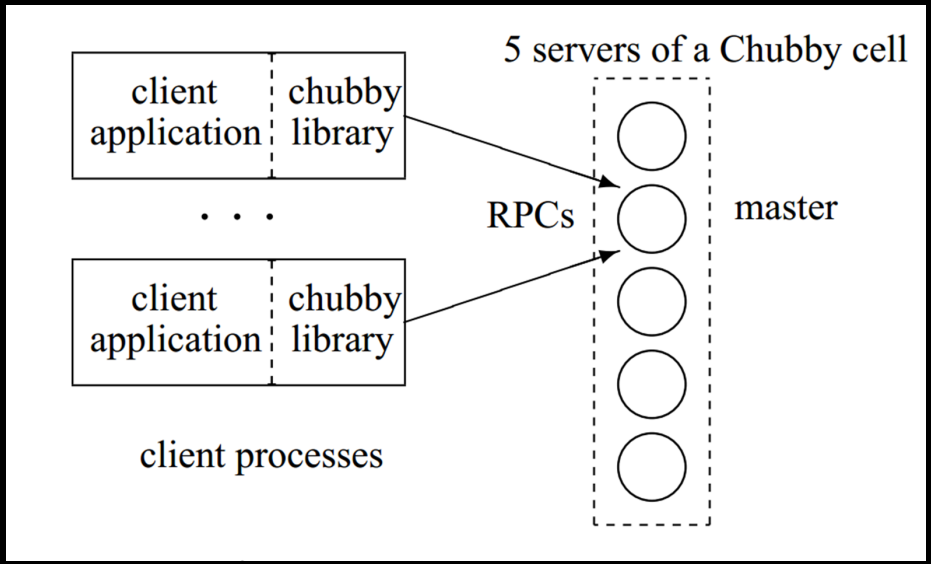
# Overview of Chubby
## Why Lock Services?

- Maintain program structure and communication patterns
- Mechanism for advertising results
- Easier to persuade programmers to use it
- Reduce number of client servers needed to make progress

# Overview of Chubby
## What is This Paper About?

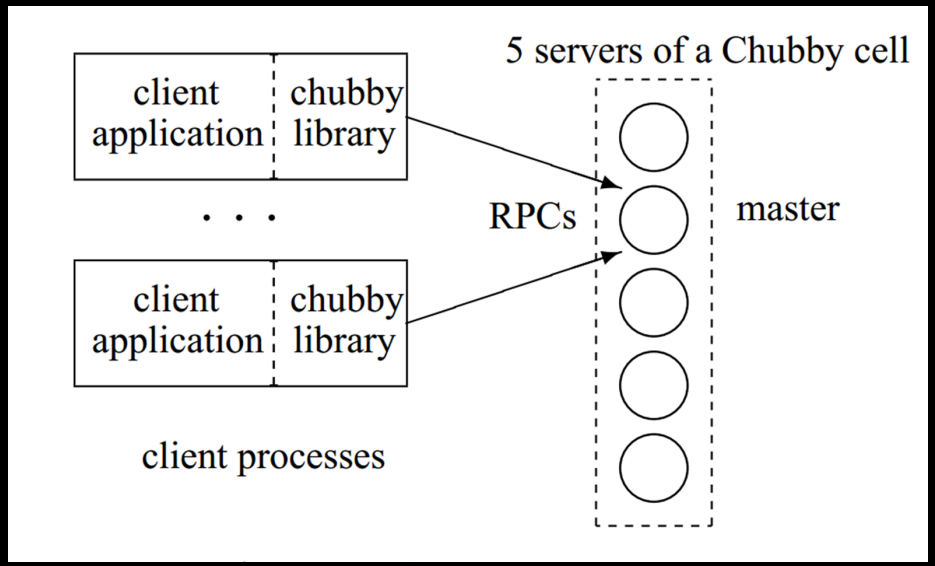
"Building Chubby was an ***engineering effort***... it was not research. We claim no new algorithms or techniques. The purpose of this paper is to describe what we did and why, rather than to advocate it" Chubby paper, OSDI 2006

# System Architecture (1)



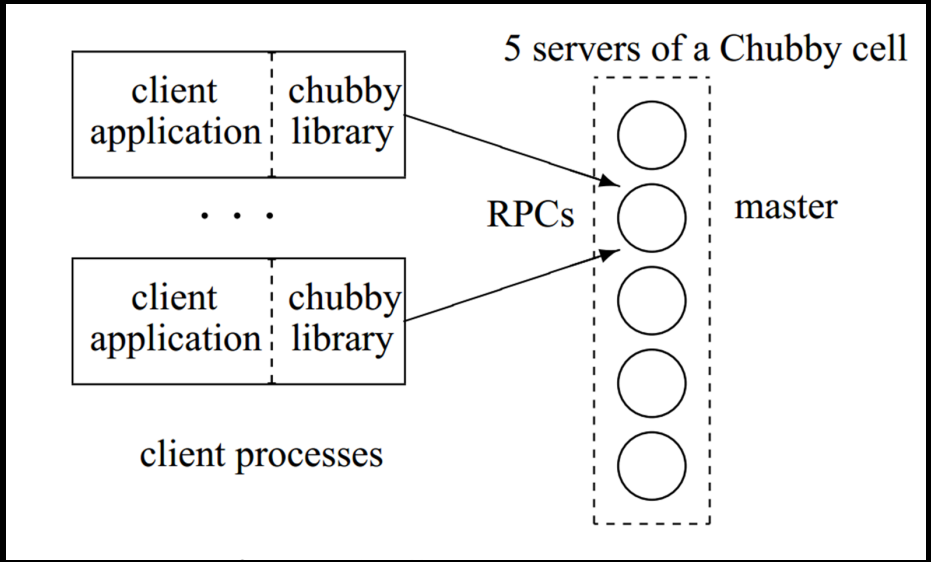

- A chubby cell consists of a small set of servers (replicas)
- A master is elected from the replicas via distributed consensus protocol (probably Paxos)
- Client talks to the master via the chubby library and clients discover master by talking to any replica

# System Architecture (2)




- Replicas maintain copies of a simple database
- Clients send read/write requests only to the master
- The master propagates updates from write to replicas via consensus protocol. Replies after the write reaches a majority of replicas.
- The master can satisfy the read request alone.

# System Architecture (3)




- If master dies, a new one will be elected, but only ***after*** master leases expire
- If a replica dies and does not recover for a long time (a few hours) $\rightarrow$ A new replica will be selected $\rightarrow$ DNS list will be updated $\rightarrow$ Master notice the updates in DNS and Integrate the new replica into the cell

# Client Interface (1)

- Interface presents as distributed file system
  - Many functionalities are deleted
- Clients can open/close/read/write files
  - Reads and writes are whole-file
  - Advisory read and write locks are supported
  - Clients can register for notification of file update

# Client Interface (2)

- Chubby supports a strict tree of files and directories
  - Regard these file and directories as locks with extra contents (e.g., identity and location of a primary)
- Example: /ls/foo/wombat/pouch
  - 1st component (ls): lock service (common to all names)
  - 2nd component (foo): the chubby cell
  - The rest: name inside the cell which is referred to as ***node***)

# Client Interface (3)

- Permanent node Versus ephemeral node
  - Ephemeral nodes are deleted if no clients open them.
- Meta-data for nodes
  - Three names of Access Control Lists (ACL)
  - Four monotonically increasing 64-bit numbers
  - A 64-bit file-content checksum

# Client Interface (4)

Clients open a node to obtain a ***handler*** which contains:

- Check digits encoded in file handler
  - Used to prevent clients from forging handler
- Sequence number
  - Used by master to tell whether the handler are created by previous master
- Mode information
  - Used by new master to learn the mode of the handler

# Lock Service

- Each Chubby file and directory can act as a reader or writer lock
  - One client may hold the lock in exclusive (writer) mode.
  - Any number of clients hold the lock in shared (reader) mode.
- To acquire a lock in either mode, you must have write permission.

# Distributed Lock Service
## Problem and Solutions

- Problem: delayed communication
  - A holds a lock L, issues request W, then fails
  - B acquires L (because A fails), performs actions
  - W arrives (out-of-order) after B's actions
- Solution 1 (Lock-Delay):
  - Chubby will not allocate lock for a period.
- Solution 2 (Sequencer):
  - A lock holder obtain a sequencer from Chubby
  - It attaches the sequencer to any requests that it sends to other servers (e.g., Bigtable)
  - The other servers can verify the sequencer information

# Events





Clients can subscribe to various events

- File contents modified
- Master failed over
- Child node added, removed, modified
- Handle becomes invalid
- Lock acquired (rarely used)
- Locks are conflicting (rarely used)

# Chubby Lock Service
## Example: Primary Election

- Potential primaries open the lock file acquire locks
- Only one succeeds and becomes the primary.
- Primary writes its identity into the lock file
- Others can learn who is the primary by reading the lock file
- Ideally, sequence number can be used

# Caching

- Client can cache the file data, meta-data, handlers and lock locally with a cache lease.
- Master will invalidate cached copies upon a write request
- Node will be treated as uncachable until master knows that the invalidation is successful

# Sessions and KeepAlives (1)

A session between a client and Chubby exists for some time and is maintained by periodic handshakes called KeepAlives:

- A client sends a KeepAlive request to a master
- The master will block KeepAlive until close to the timeout
- A master responds by a KeepAlives response and extend the session lease
- Immediately after getting the keep-alive response, the client sends another request for extension

# Sessions and KeepAlives (2)

- Clients maintain a local timer for estimating the session timeouts (time is not perfectly synchronized)
- If local timer runs out, wait for a 45s grace period before ending the session
  - Used when a master fails over happens

# Master Fail-overs (1)

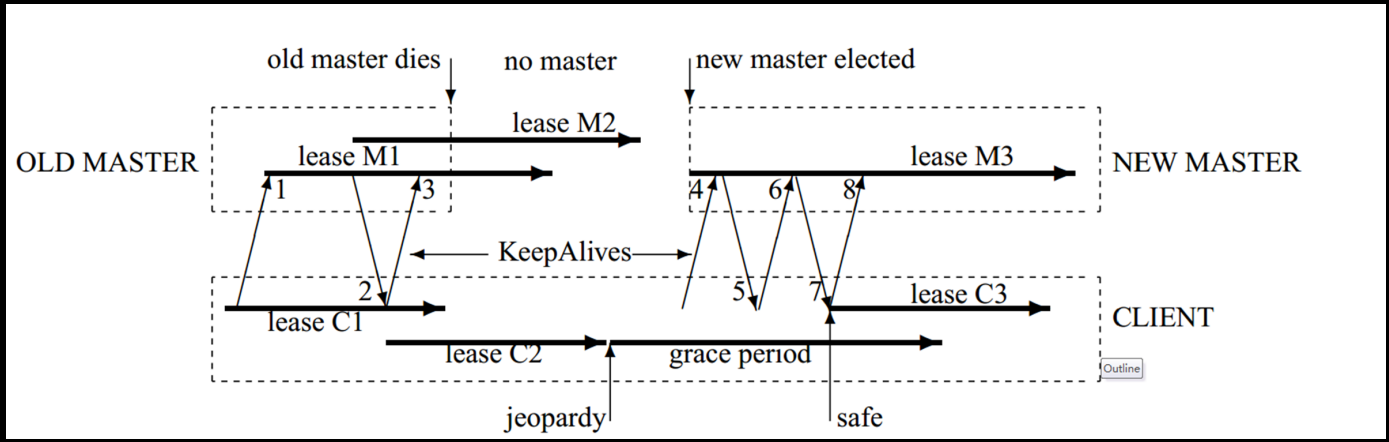
- In-memory state discarded
- Lease timer for master stops
- Fast master election
  - client reconnect before lease expires
- Slow master election
  - clients flush cache, enter grace period
- New master reconstruct the in-memory state of previous master

# Master Fail-overs (2)

For the new master to take over:

- Pick new epoch number
- Respond only to master location requests
- Reconstruct in-memory state
- Respond to KeepAlives
- Emit fail-over events
- Wait for acknowledgements / session expire
- Allow all operations to proceed
- Allow clients to use handles created before fail-over
- Delete ephemeral files without open handles after an interval

# Master Fail-overs (3)



# Proxy Server

- Proxies pass write request to cell
- Proxies handle KeepAlives and reads
  - Write does not happen frequently in Chubby
- Disadvantage:
  - Additional communication overhead
  - Increase probability of unavailability

# Partitioning

- Namespace of a cell partitioned between servers
- N partitions, each with master and replicas
- Little cross-partition communication
- Reduce write and read load on a given partitions

# More on Fail-over Problems

- Master writes sessions to Database when created
  - Too heavy overhead
- New Design 1: store session at first write operations and read-only sessions are written to at random.
  - Young read-only sessions may be discarded in fail-overs
- New Design 2: Do not store sessions at all
  - New master waits worst-case lease timeout before operations.
- Fail-overs on Proxies: Allow proxies to change the session that a lock is associated with
  - Master do not relinquishing them or ephemeral file handles until new proxies can claim them.

# Abusive Clients

- Lack of aggressive caching
  - Indefinite loops to open absent files or repeatedly open file handles
- Lack of quotas
  - Put limit on file size introduced
  - Usage of other file system are encouraged
- Publish/subscribe
  - Event mechanisms is not suitable for most publish/subscribe functionalities

# Lessons Leanrt

- Developers rarely consider availability
  - Should plan for short Chubby outages
- Fine-grained locking not essential
  - Can write application-level fine-grained locking
- Poor API choices
  - Handles acquiring locks cannot be shared
- Communication use affects transport protocols
  - Send KeepAlives by UDP for timeliness (sometimes)

# Summary of Chubby

- Distributed lock service intended for coarse grained synchronization of activities
- Design based on well-known ideas
- Primary internal name service
- Repository for files requiring high availability



# Q&A