The Chubby lock service for loosely-coupled distributed systems (Lecture 22, cs262a)

Ion Stoica,
UC Berkeley
November 7, 2016

Today's Paper

The Chubby lock service for loosely-coupled distributed systems,

Mike Burrows, OSDI'06

(https://www.usenix.org/event/osdi06/tech/full_papers/burrows/burrows.pdf)

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.”

Introduction

What is Chubby?

- Lock service in a loosely-coupled distributed system (e.g., 10K 4-processor machines connected by 1Gbps Ethernet)
- Client interface similar to whole-file advisory locks with notification of various events (e.g., file modifications)
- Primary goals: reliability, availability, easy-to-understand semantics

How is it used?

- Used in Google: GFS, Bigtable, etc.
- Elect leaders, store small amount of meta-data, as the root of the distributed data structures

Design

Lock service, instead of consensus library

Serve small files

Support large-scale concurrent file viewing

Event notification mechanism

Caching of files (consistent caching)

Security, including access control

Design Decisions

Lock service vs. consensus (Paxos) library

Advantages:

- Maintain program structure, communication patterns
- Can support notification mechanism
- Smaller # of nodes (servers(needed to make progress

Advisory instead of mandatory locks (why?):

- Holding a lock called F neither is necessary to access the file F, nor prevents other clients from doing so

Design Decisions

Coarse vs. fine-grained locks

Advantages of coarse-grained locks

- Less load on lock server
- Less delay when lock server fails
- Less lock servers and availability required

Advantages of fine-grained locks

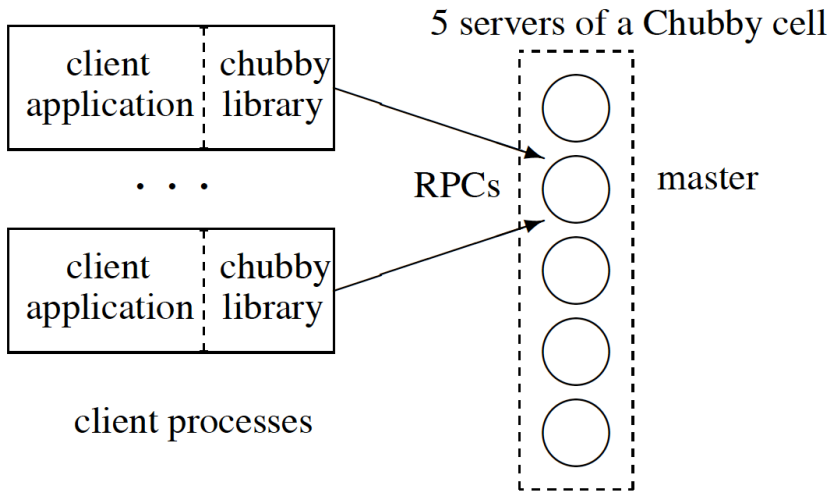
- More lock server load
- If needed, could be implemented on client side

System Structure

Chubby cell: a small number of replicas (e.g., 5)

Master is selected using a consensus protocol (e.g., Paxos)

- Master gets a lease of several seconds
- If master fails, it's lease expires and a new one is selected



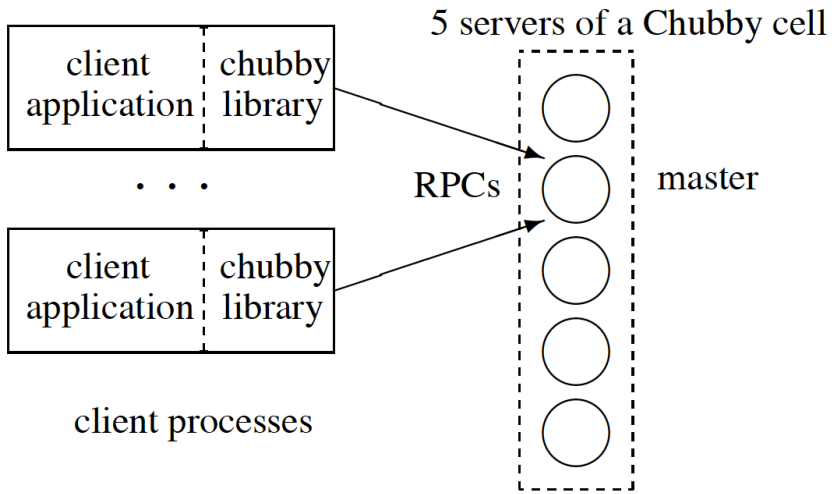
System Structure

Clients

- Send reads/writes only to the master
- Communicates with master via a chubby library

Every replica server

- Is listed in DNS
- Direct clients to master
- Maintain copies of a simple database



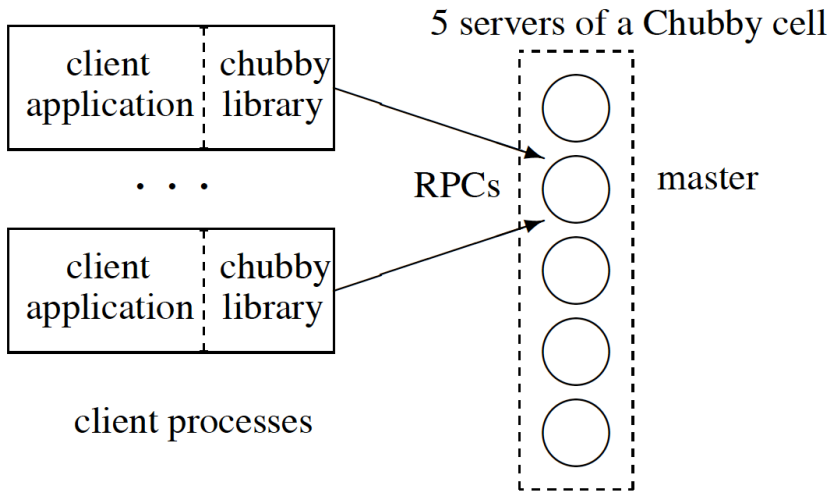
Read and Writes

Write

- Master propagates write to replica Replies after the write reaches a majority (e.g., quorum)

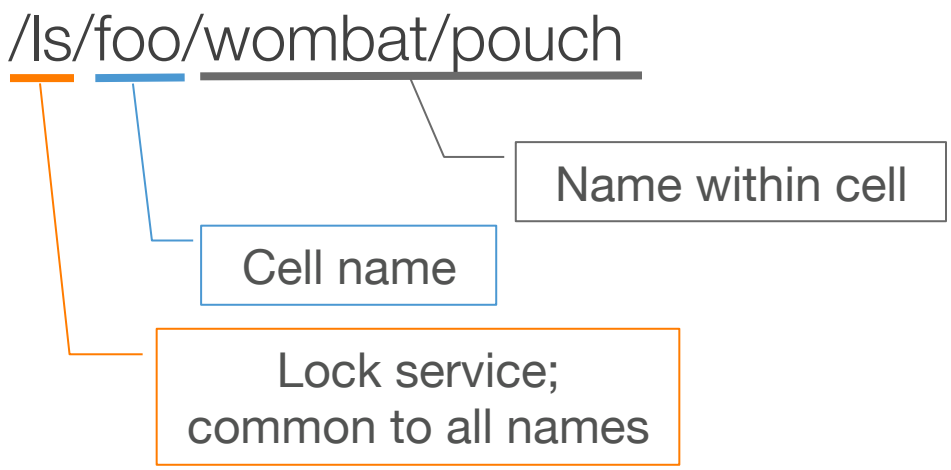
Read

- Mater replies directly, as it has most up to date state



Simple UNIX-like File System Interface

Bare bone file & directory structure



Simple UNIX-like File System Interface

Bare bone file & directory structure

/ls/foo/wombat/pouch

Does not support, maintain, or reveal

- Moving files
- Path-dependent permission semantics
- Directory modified times, files last-access times

Nodes

Node: a file or directory

- Any node can act as an advisory reader/writer lock

A node may be either permanent or ephemeral

- Ephemeral used as temporary files, e.g., indicate a client is alive

Metadata

- Three names of ACLs (R/W/change ACL name)
  - Authentication build into ROC
- 64-bit file content checksum

Locks

Any node can act as lock (shared or exclusive)

Advisory (vs. mandatory)

- Protect resources at remote services
- No value in extra guards by mandatory locks

Write permission needed to acquire

- Prevents unprivileged reader blocking progress

Locks and Sequences

Potential lock problems in distributed systems

- 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: backward compatible

- Lock server will prevent other clients from getting the lock if a lock become inaccessible or the holder has failed
- Lock-delay period can be specified by clients

Locks and Sequences

Potential lock problems in distributed systems

- 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 2: sequencer

- A lock holder can obtain a sequencer from Chubby
- It attaches the sequencer to any requests that it sends to other servers
- The other servers can verify the sequencer information

Design: Events

Client subscribes when creating handle

Delivered async via up-call from client library

Event types

- File contents modified
- Child node added / removed / modified
- Chubby master failed over
- Handle / lock have become invalid
- Lock acquired / conflicting lock request (rarely used)

Design: API

Open() (only call using named node)

- how handle will be used (access checks here)
- events to subscribe to
- lock-delay
- whether new file/dir should be created

Close() vs. Poison()

Other ops:

- GetContentsAndStat(), SetContents(), Delete(), Acquire(), TryAcquire(), Release(), GetSequencer(), SetSequencer(), CheckSequencer()

Caching

Client caches file data, node meta-data

- Write-through held in memory

Strict consistency: easy to understand

- Lease based
- Master will invalidate cached copies upon a write request

Handles and locks cached as well

- Event informs client of conflicting lock request

Sessions and Keep-Alives

A client sends keep-alive requests to a master

A master responds by a keep-alive response

Immediately after getting the keep-alive response, the client sends another request for extension

The master will block keep-alives until close the expiration of a session

Extension is default to 12s

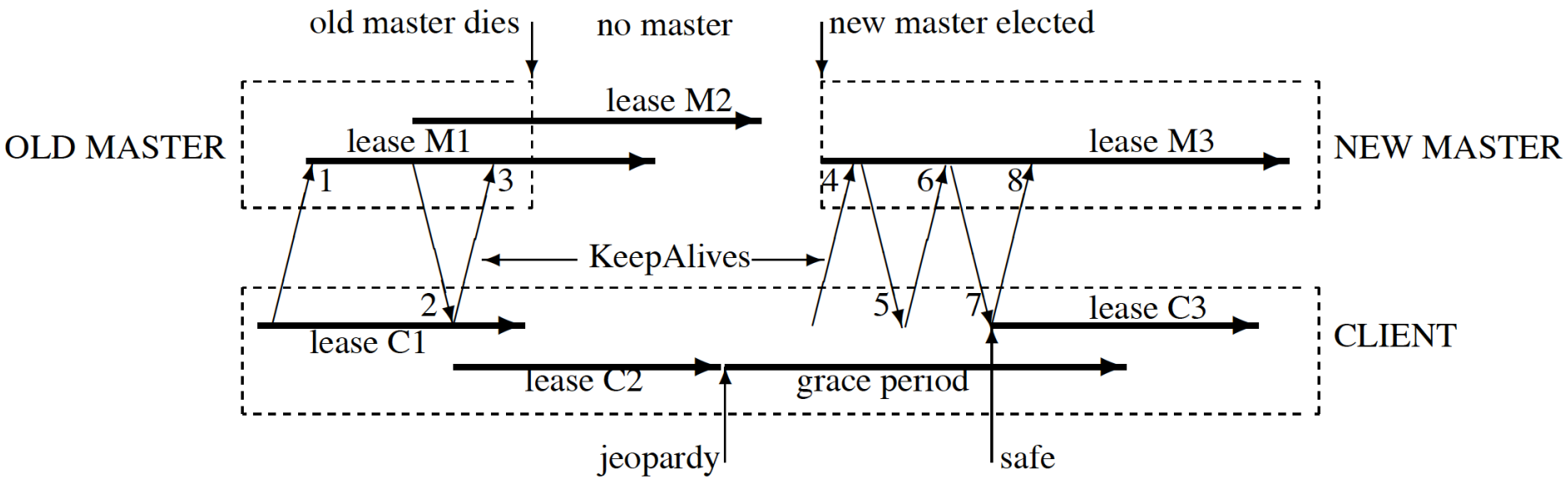
Sessions and Master Fail-overs

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

- Happens when a master fails over

Example


old master dies
no master
new master elected
OLD MASTER
lease M2
lease M1
lease M3
NEW MASTER
1
3
4
6
8
KeepAlives
2
5
7
lease C3
lease C1
CLIENT
lease C2
grace period
jeopardy
safe

Other Details

Database implementation

- A simple database with write ahead logging and snapshotting
- Switched from BerkeleyDB to bare-bone internally-developed DB (why?)

Backup:

- Write a snapshot to a GFS server in a different building

Mirroring files across multiple cells

- Configuration files (e.g., locations of other services, access control lists, etc.)

Why scaling is important?

Clients connect to a single instance of master in a cell

- Much more client processes that number of machines

Existing mechanisms:

- More Chubby cells
- Increase lease time from 12s to 60s to reduce KeepAlive messages (dominant requests in experiments)
- Client caches

New Mechanisms

Proxies:

- Handle KeepAlive and read requests, pass write requests to the master
- Reduce traffic but reduce availability

Partitioning: partition name space

- A master handles nodes with hash(name) mod N == id
- Limited cross-partition messages

Cell outages Stats

61 outages for all cells in a few weeks

- 52 outages were under 30s, incur under 30s delay in applications

Remaining 9 outages

- Network maintenance (4)
- Suspected network connectivity problems (2)
- Software errors (2)
- Overload (1)

Java Clients

Google systems are mostly written in C++

- Chubby client library is complex (7000 lines)
- Hard to port to Java and maintain in Java

JNI is inefficient

Java users run a protocol-conversion server with a simple RPC protocol

Don’t know how to avoid running this additional server

As a Name Service

Chubby's most popular use

For availability, need to set DNS TTL short, but often overwhelms DNS server

- Must poll each DNS entry: $O(N^2)$
- Chubby is invalidation based
- Besides KeepAlives, no need to poll

Abusive Clients

Review before clients can use shared Chubby cells

Problem cases encountered:

- Repeated open/close calls for polling a file
  - Cache file handles aggressively
- Lack of quotas
  - 256Kbytes file size limit
- Publish/subscribe system on Chubby? inefficient

Lesson Learned

Developers rarely consider availability

- Should plan for short Chubby outages
- Crashed applications on failover event

Fine grained locking not essential

API mostly good, but one bad chose stands out

- Handles acquiring locks cannot be shared

RPC affects transport protocols

- Forced to send KeepAlives by UDP (why?)

Summary

Distributed Lock Service

- Coarse-grained synchronization for Google's distributed systems
- Primarily used as internal name service
- Repository for files requiring high availability

Well-known techniques

- Distributed consensus, caching, notofications

UNIX-like file system interface