The Chubby lock service for loosely-coupled distributed systems

Mike Burrows, Google Inc.

Presented By- Rahul Malviya

Outline

- Paxos Algorithm
- The Chubby Lock Service

What is Paxos

- Help to implementing a fault-tolerant distributed system, provide *distributed consensus in a network of several processors*

- a family of algorithms
  - cheap Paxos, fast Paxos, generalized Paxos, Byzantine Paxos...

Why Paxos

- *Considered the most effective in the field*
- all working protocols for asynchronous consensus we have so far encountered have Paxos at their core.
- Raised from intuitive and reasonable reasoning

Paxos …

- Three roles in the Paxos protocol
  - *Proposer*
  - *Acceptor*
  - *Learner*
- Goal: consensus

How…

- The Safety properties
  - A value that has been proposed may be chosen
  - Only a single value is chosen
  - A leaner never learns that a value has been chosen unless it actually has been chosen

Reasoning

- P1: An acceptor must accept the first proposal that it receives.

- Keeping track of proposals:
  - unique numbers
  - a proposal is pairs: (counter, value)
- A value has been chosen only if:
  - A single proposal has been accepted by a majority.

Reasoning

- We can allow multiple proposals to be chosen
- But must guarantee that all chosen proposals have the same value:

- P2: If a proposal with value *v* is chosen, then every higher-numbered proposal that is chosen has value *v*.

- P2 guarantees the safety property of consensus.

Reasoning

- P2a: If a proposal with value v is chosen, then every higher-numbered proposal accepted by any acceptor has value v.
- P2b: If a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v.

Reasoning

- P2c: If a proposal (n, v) is issued, then there exists a majority S of acceptors, such that:
  - **Either** no acceptor in *S* has accepted any proposal numbered less than *n*
  - **Or** *v* is the value of the highest-numbered proposal among all proposals numbered less than *n* accepted by the acceptors in *S*.

- *P2c* → *P2b* → P2a → P2

Algorithm

- **Phase 1a: *Prepare***
  - Send Msg: Prepare n
- **Phase 1b: *Promise***
  - **For** *Acceptor,* **if n > previous proposal** number, send back the value.
- **Phase 2a: *Accept!***
  - **proposer fix the value**
- **Phase 2b: *Accepted***
  - *Acceptor* **fix the value, tell the learner**

The Chubby Lock Service（1）

Chubby's Design

- Introduction
- System structure
- File, dirs, handles
- Locks and sequencers
- Events
- API
- Caching

Basic Idea About Chubby

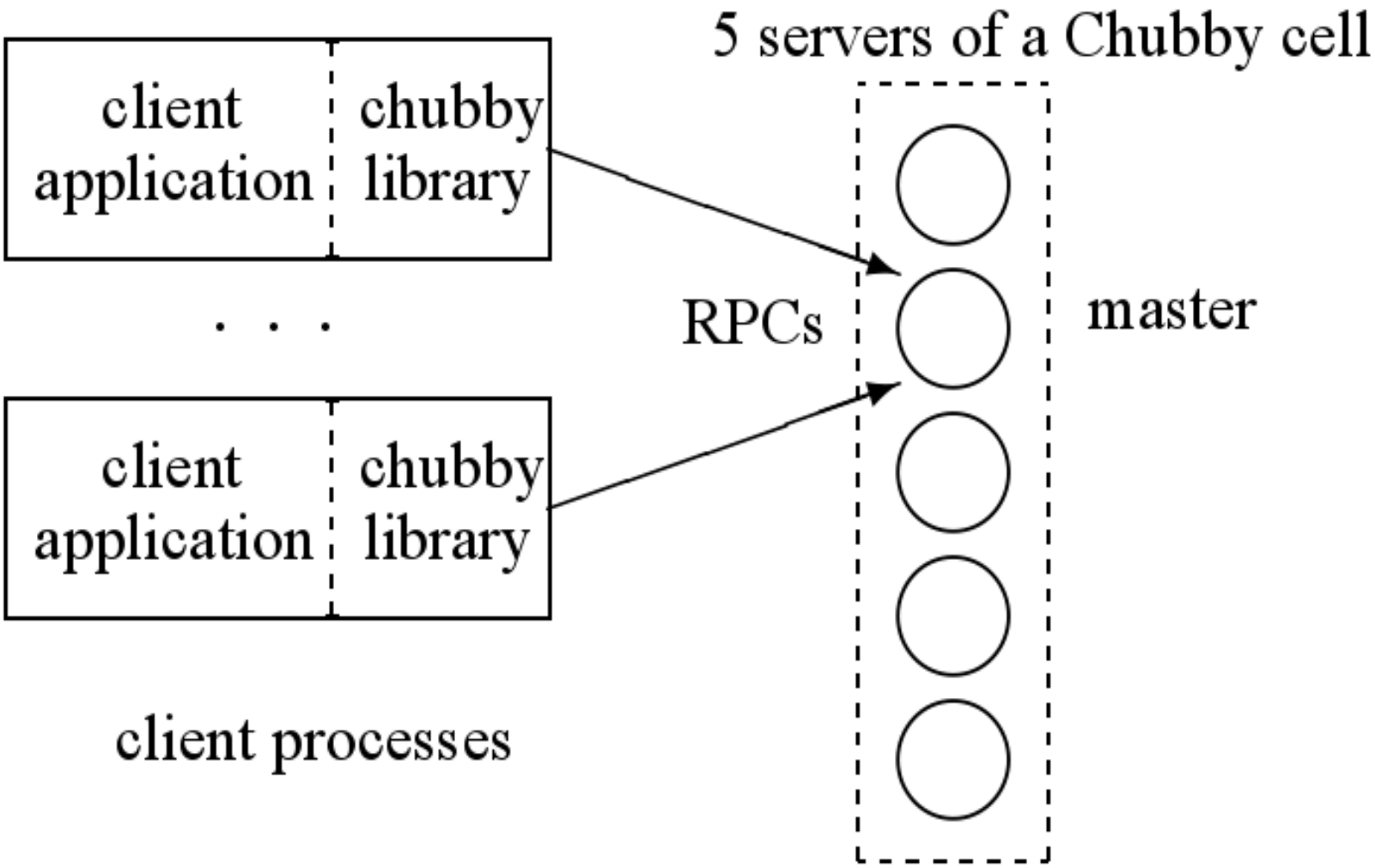
- Chubby lock service, is intended to provide coarse-grained locking as well as reliable (though low-volume) storage for a loosely-coupled distributed system.
- Chubby provides an interface much like a distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance.

Introduction

- It is intended for use within a loosely-coupled distributed system consisting of moderately large numbers of small machines connected by a high-speed network.

- The purpose of the lock service is to allow its clients to synchronize their activities and to agree on basic information about their environment. The primary goals included reliability, availability to a moderately large set of clients, and throughput and storage capacity were considered secondary.

System structure



System Structure

- Chubby has two main components that communicate via RPC: a server, and a library that client applications link against
- A Chubby cell consists of a small set of servers (typically five) known as *replicas*.
- The replicas use a distributed consensus protocol to elect a *master;* the master must obtain votes from a majority of the replicas, plus promises that those replicas will not elect a different master for an interval of a few seconds known as the *master lease*

Files, dirs, handles

- File interface similar to, but simpler than UNIX
  - /ls/foo/wombat/pouch
- the name space contains files and directories called nodes
- each node has various meta-data

Files, dirs, handles (cont.)

- clients open node to obtain handles
- handles ~ UNIX file descriptors
- handle include
  - Check digits – prevent client guess handle
  - Sequence number
  - Mode information – use to recreate the lock state when the master changes

Locks and sequencers

- Chubby uses a file or directory to act as a reader-writer lock
- One client may hold the lock in exclusive (writer) mode.
- Any number of clients hold the lock in shared (reader) mode.
- Locks are advisory - only conflict with other attempts to acquire the same lock

Locks and sequencers

- Locking in Distributed is complex
- Virtual synchrony or virtual time – costly
- Chubby - Only interactions which use locks are numbered
- Sequencer - a byte string describing the state of the lock after acquisition
- A lock requests a sequencer
- A client passes the sequencer to the server for protected progress

Events

- Client can subscribe to some events when a handle is created.
- Delivered after the corresponding action has taken place
- Events include
  - file contents modified
  - child node added, removed, or modified
  - chubby master failed over
  - handle has become invalid
  - lock acquired by others
  - conflicting lock requested from another client

API

- Handled are created by Open()
- destroyed by Close()
- GetContentsAndStat(), GetStat(), ReadDir()
- SetContents(), SetACL()
- Delete()
- Acquire(), TryAcquire(), Release()
- GetSequencer(), SetSequencer(), CheckSequencer()

Caching

- Client caches file data and meta-data
  reduce read traffic
- Handle and lock can also be cached
- Master keeps a list of which clients might be caching
- Invalidations keep consistent
- Clients see consistent data or error

The Chubby Lock Service（2）

Client Sessions

- **Sessions** maintained between client and server
  - Keep-alive messages required to maintain session every few seconds
- If session is lost, server releases any client-held handles.
- What if master is late with next keep-alive?
  - Client has its own (longer) timeout to detect server failure

Master Failure

- If client does not hear back about keep-alive in *local lease timeout*, session is **in jeopardy**
  - Clear local cache
  - Wait for "grace period" (about 45 seconds)
  - Continue attempt to contact master
- Successful attempt => ok; jeopardy over
- Failed attempt => session assumed lost

Master Failure

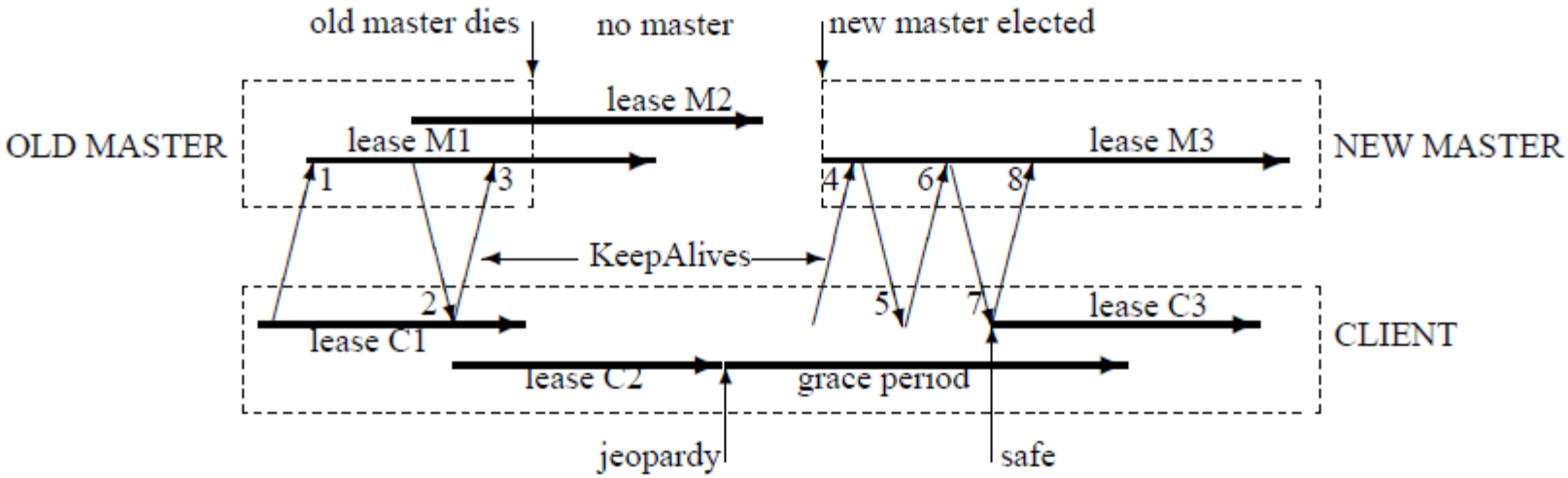


Figure 2: The role of the grace period in master fail-over

Scalability

- Arbitrary number of Chubby cells.
- System increases lease times from 12 sec up to 60 sec under heavy load
- Chubby clients cache file data, meta-data, the absence of files, and open handles.
- Data is small – all held in RAM (as well as disk)
- * Use proxies to save KeepAlive and reads traffic.
- * Use partitioning

time since last fail-over	18 days
fail-over duration	14s
active clients (direct)	22k
additional proxied clients	32k
files open	12k
naming-related	60%
client-is-caching-file entries	230k
distinct files cached	24k
names negatively cached	32k
exclusive locks	1k
shared locks	0
stored directories	8k
ephemeral	0.1%
stored files	22k
0-1k bytes	90%
1k-10k bytes	10%
> 10k bytes	0.2%
naming-related	46%
mirrored ACLs & config info	27%
GFS and Bigtable meta-data	11%
ephemeral	3%
RPC rate	1-2k/s
KeepAlive	93%
GetStat	2%
Open	1%
CreateSession	1%
GetContentsAndStat	0.4%
SetContents	680ppm
Acquire	31ppm

REFERENCES

- Paxos
  - [Chandra, et al.,2007] T. D. Chandra, R. Griesemer, and J. Redstone, "Paxos made live: an engineering perspective," in *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. Portland, Oregon, USA: ACM, 2007, pp. 398-407. http://labs.google.com/papers/paxos_made_live.html
  - [Lamport,2001] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, pp. 18-25, 2001.
  - http://en.wikipedia.org/wiki/Paxos_algorithm
- [Burrows,2006] M. Burrows, "The Chubby Lock Service for Loosely-Coupled Distributed Systems," presented at OSDI'06: Seventh Symposium on Operating System Design and Implementation, Seattle, WA, 2006.
- Zookeeper
  - http://zookeeper.wiki.sourceforge.net
  - http://developer.yahoo.com/blogs/hadoop/2008/03/intro-to-zookeeper-video.html

Thank You !