The Chubby lock service for loosely-coupled distributed systems

Presented by Petko Nikolov
Cornell University
3/12/09

Before Chubby Came About…

- Wide range of distributed systems
- Clients in the 10,000s
- How to do primary election?
  – Ad hoc (no harm from duplicated work)
  – Operator intervention (correctness essential)
- Disorganized
- Costly
- Low availability

Motivation

- Need for service that provides
  - synchronization (leader election, shared env. info.)
  - reliability
  - availability
  - easy-to-understand semantics
  - performance, throughput, latency only secondary
- NOT research

Outline

- Primary Election
  - Paxos
- Design
- Use and Observations
- Related Work

Primary Election

- Distributed consensus problem
- Asynchronous communication
  - loss, delay, reordering
- FLP impossibility result
- Solution: Paxos protocol

Paxos: Problem

- Collection of processes proposing values
  - only proposed value may be chosen
  - only single value chosen
  - learn of chosen value only when it has been
- Proposers, acceptors, learners
- Asynchronous, non-Byzantine model
  - arbitrary speeds, fail by stopping, restart
  - messages not corrupted

Paxos: Algorithm

Phase 1

(a) Proposer sends *prepare* request with #n

(b) Acceptor: if n > # of any other *prepare* it has replied to, respond with promise.

Phase 2

(a) If majority reply, proposer sends accept with value v

(b) Acceptor accepts unless it responded to *prepare* with # higher than n

Paxos: Algorithm

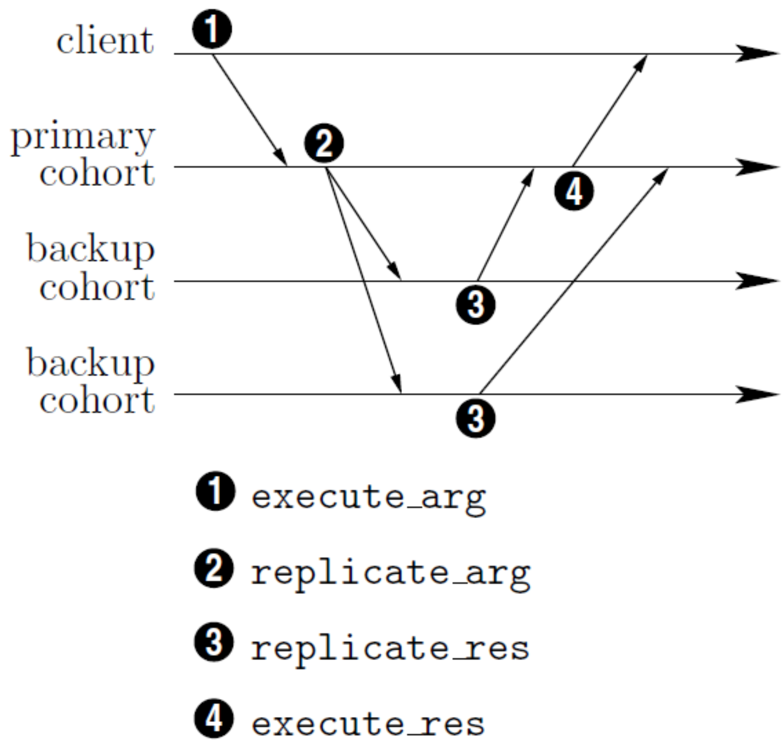

Figure 2: Messages sent during normal-case operation, when no cohorts fail or join the system.

Paxos: Algorithm

- Learning of chosen value
  - distinguished learner optimization
    - has pitfalls
- Making progress
  - distinguished proposer
- Usually, every process plays all roles
  - *primary* as distinguished proposer and learner

Paxos: State Machines

- Replicated state machine
  - same state if same sequence of ops. performed
- Client sends requests to server
  - replicated with Paxos
- Paxos used to agree on order of client ops.
  - can have failures / more than 1 master
  - Paxos guarantees only 1 value chosen & replicated

Paxos: View Change

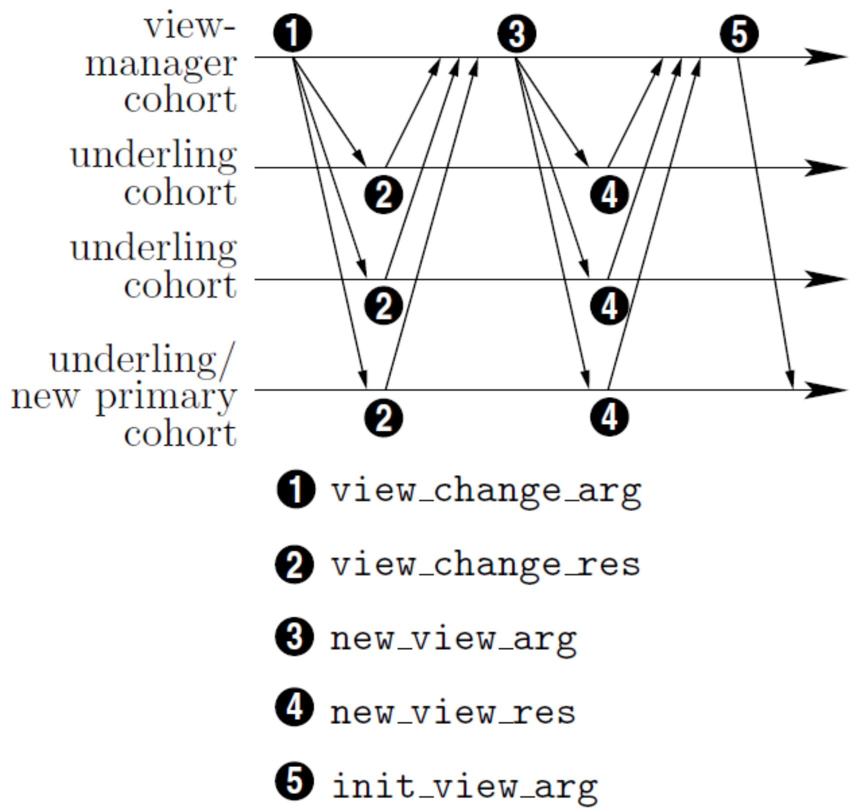

Figure 4: Messages sent during a view change.

Design

- Lock service (and not consensus library)
- Serve small files
- Support large-scale concurrent file viewing
- Event notification mechanism
- Caching of files (consistent caching)
- Security (access control)
- Course-grained locks

Design: Rationale

- Lock service vs. Paxos library
- Advantages
  - maintain program structure, comm. patters
  - mechanism for advertising results
  - persuading programmers to use it
  - reduce # of client servers needed to make progress

Design: Rationale

- Course-grained locks
  - less load on lock server
  - less delay when lock server fails
  - should survive lock server failures
  - less lock servers and availability required
- Fine-grained locks
  - heavier lock server load, more client stalling on fail
  - can be implemented on client side

Design: System Structure

- Two main components:
  - server (Chubby cell)
  - client library
  - communicate via RPC
- Proxy
  - optional
  - more on this later

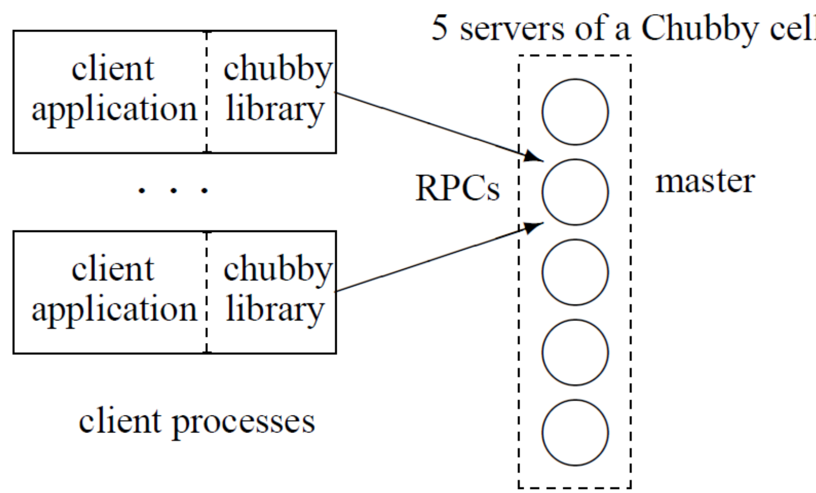

Figure 1: System structure

Design: Chubby Cell

- Set of replicas (typically 5)
- Use Paxos to elect master
  - promise not to elect new master for some time (master lease)
- Maintain copies of simple database
- Writes satisfied by majority quorum
- Reads satisfied by master alone
- Replacement system for failed replicas

Design: Chubby Clients

- Link against library
- Master location requests to replicas
- All requests sent directly to master

Design: Files, Dirs, Handles

- FS interface
  - /ls/cs6464-cell/lab2/test
  - specialized API
  - also via interface used by GFS
- Does not support/maintain/reveal
  - moving files
  - path-dependent permission semantics
  - dir modified times / file last-access times

Design: Nodes

- permanent vs. ephemeral
- Metadata
  - three names of ACLs (R/W/change ACL name)
    - authentication built into RPC
  - 4 monotonically increasing 64-bit numbers
    - instance, content generation, lock generation, ACL gen.
  - 64-bit file-content checksum

Design: Handles

- Analogous to UNIX file descriptors
- Check digits
  - prevent client creating/guessing handles
- Support for use across master changes
  - sequence number
  - mode information for recreating state

Design: Locks and Sequencers

- Any node can act as lock (shared or exclusive)
- Advisory (vs. mandatory)
  - protect resources at remote services
  - debugging / admin. purposes
  - no value in extra guards by mandatory locks
- Write permission needed to acquire
  - prevents unprivileged reader blocking progress

Design: Locks and Sequencers

- Complex in async environment
- Use sequence #'s in interactions using locks
- Sequencer
  - opaque byte-string
  - state of lock immediately after acquisition
  - passed by client to servers, servers validate
- Alternative: lock-delay

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()

Design: API

- Primary election example
- Candidates attempt to open lock file / get lock
  - winner writes identity with SetContents()
  - replicas find out with GetContentsAndStat(), possibly after file-modification event
- Primary obtains sequencer (GetSequencer())

Design: Sessions and KeepAlives

- Session maintained through KeepAlives
- Handles, locks, cached data remain valid
  - client must acknowledge invalidation messages
- Terminated explicitly, or after lease timeout
- Lease timeout advanced when
  - session created
  - master fail-over occurs
  - master responds to KeepAlive RPC

Design: Sessions and KeepAlives

- Master responds close to lease timeout
- Client sends another KeepAlive immediately

Design: Sessions and KeepAlives

- Handles, locks, cached data remain valid
  - client must acknowledge invalidation messages
- Cache invalidations piggybacked on KeepAlive
  - client must invalidate to maintain session
  - RPC's flow from client to master
  - allows operation through firewalls

Design: Sessions and KeepAlives

- Client maintains local lease timeout
  - conservative approximation
  - must assume known restrictions on clock skew
- When local lease expires
  - disable cache
  - session in *jeopardy*, client waits in *grace* period
  - cache enabled on reconnect
- Application informed about session changes

Design: Caching

- Client caches file data, node meta-data
  - write-through held in memory
- Invalidation
  - master keeps list of what clients may have cached
  - writes block, master sends invalidations
  - clients flush changed data, ack. with KeepAlive
  - data *uncachable* until invalidation acked
    - allows reads to happen without delay

Design: Caching

- Invalidates data but does not update
  - updating arbitrarily unefficient
- Strict vs. weak consistency
  - weaker models harder to use for programmers
  - do not want to alter preexisting comm. protocols
- Handles and locks cached as well
  - event informs client of conflicting lock request
- Absence of files cached

Design: Fail-overs

- In-memory state discarded
  - sessions, handles, locks, etc.
- Lease timer “stops”
- Quick re-election
  - client reconnect before leases expire
- Slow re-election
  - clients disable cache, enter grace period
  - allows sessions across fail-overs

Design: Fail-overs

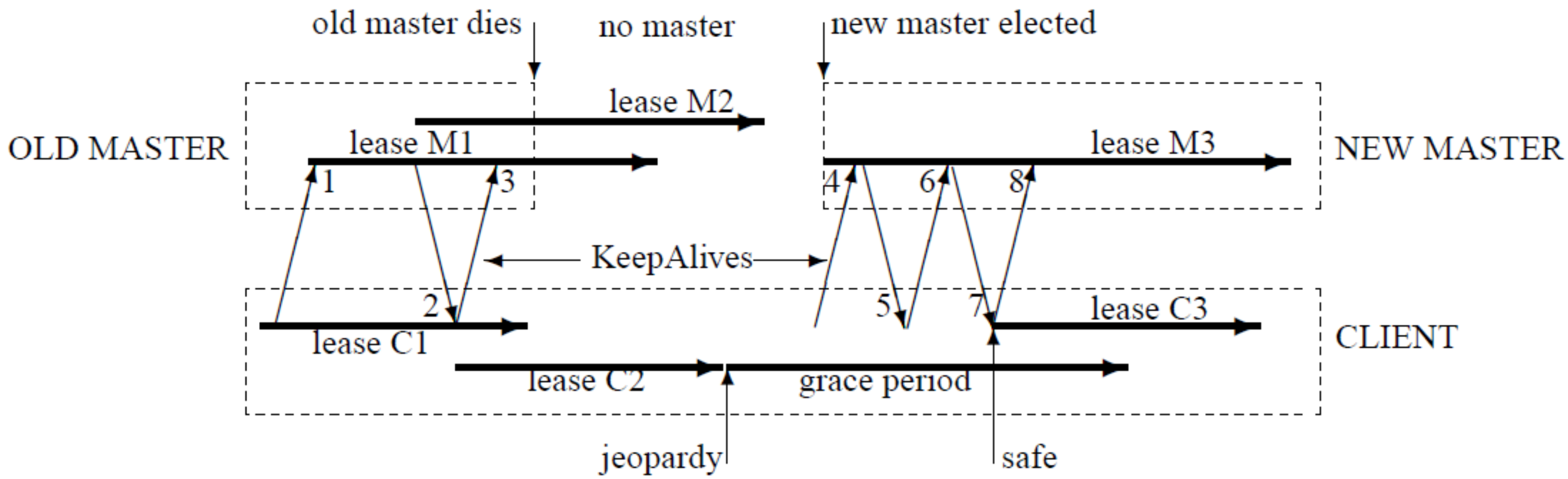

Figure 2: The role of the grace period in master fail-over

Design: Fail-overs

Steps of newly-elected master:

1. Pick new epoch number
2. Respond only to master location requests
3. Build in-memory state for sessions / locks from DB
4. Respond to KeepAlives
5. Emit fail-over events to caches
6. Wait for acknowledgements / session expire
7. Allow all operations to proceed

Design: Fail-overs

Steps of newly-elected master (cont'd):

8. Handle created pre-fail-over used
   - master recreates in memory, honors call
   - if closed, record that in memory

10. Delete ephemeral files w/o open handles after an interval

- Fail-over code source of many bugs

Design: Database

- First Chubby used replicated Berkeley DB
  - with master lease added on
- Replication code was new
  - did not want to take the risk
- Implemented own simple database
  - distributed using consensus protocol

Design: Backup

- Every few hours
- Snapshot of database to GFS server
  - different building
    - building damage, cyclic dependecies
- Disaster recovery
- Initialize new replica
  - avoid load on in-service replicas

Design: Mirroring

- Collection of files mirrored across cells
- Mostly for configuration files
  - /ls/global/master mirrored to /ls/*cell*/slave
    - global cell's replicas spread around world
  - Chubby's own ACLs
  - Files advertising presence / location
  - pointers to Bigtable cells
  - etc.

Mechanisms for Scaling

- Clients individual processes (not machines)
  - observed 90,000 clients for a single master
- Server machines identical to client ones
- Most effective scaling: reduce communication
- Regulate # of Chubby cells
- Increase lease time
- Caching
- Protocol-conversion servers

Scaling: Proxies

- Proxies pass requests from clients to cell
- Can handle KeepAlives and reads
- Not writes, but they are << 1% of workload
- KeepAlive traffic by far most dominant
- Disadvantages:
  - additional RPC for writes / first time reads
  - increased unavailability probability
  - fail-over strategy not ideal (will come back to this)

Scaling: Partitioning

- Namespace partitioned between servers
- N partitions, each with master and replicas
- Node D/C stored on P(D/C) = hash(D) mod N
  - meta-data for D may be on different partition
- Little cross-partition comm. desirable
  - permission checks
  - directory deletion
  - caching helps mitigate this

Use and Observations

- Many files for naming
- Config, ACL, meta-data common
- 10 clients use each cached file, on avg.
- Few locks held, no shared locks
- KeepAlives dominate RPC traffic

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

Use: Outages

- Sample of cells
  - 61 outages over few weeks (700 cell-days)
  - due to network congestion, maintenance, overload, errors in software, hardware, operators
- 52 outages under 30s
  - applications not significantly affected
- Few dozen cell-years of operation
  - data lost on 6 occasions (bugs & operator error)

Use: Java Clients

- Most of Google infrastructure is in C++
- Growing # of Java applications
- Googlers dislike JNI
  - would rather translate library to Java
  - maintaining it would require great expense
- Java users run protocol-conversion server
  - exports protocol similar to Chubby's client API

Use: Name Service

- Most popular use of Chubby
  - provides name service for most Google systems
- DNS uses TTL values
  - entries must be refreshed within that time
  - huge (and variable) load on DNS server
- Chubby's caching uses invalidations, no polling
  - client builds up needed entries in cache
  - name entries further grouped in batches

Use: Name Service

- Name service
  - no full consistency needed
  - reduce load with protocol-conversion server
- Chubby DNS server
  - naming data available to DNS clients
  - eases transition between names
  - accommodates browsers

Use: Fail-over Problems

- Master writes sessions to DB when created
  - start of many processes at once = overload
- DB modified – store session at first write op.
  - read-only sessions: at random on KeepAlives
  - spread out writes to DB in time
- Young read-only sessions may be “discarded”
  - may read stale data for a while after fail-over
  - very low probability

Use: Fail-over Problems

- New design – no sessions in database
  - recreate them like handles after fail-over
  - new master waits full lease time before ops.
    - little effect – very low probability
- Proxy servers can manage sessions
  - allowed to change session a lock is associated with
    - permits takeover of session by another proxy on fail
  - master gives new proxy chance to claim locks before relinquishing them

Use: Abusive Clients

- Company environment assumed
- Requests to use Chubby thoroughly reviewed
- Abuses:
  - lack of aggressive caching
    - absence of files, open file handles
  - lack of quotas
    - 256kB limit on file size introduced
    - encouraged use of appropriate storage systems
  - publish/subscribe

Use: Lessons Learned

- Developers rarely consider availability
  - should plan for short Chubby outages
  - crashed applications on fail-over event
- Fine-grained locking not essential
- Poor API choices
  - handles acquiring locks cannot be shared
- RPC use affects transport protocols
  - forced to send KeepAlives by UDP for timeliness

Related Work

Chubby

- locks, storage system, session/lease in one service
- target audience – wide range
- higher-level interface
- lost lock expensive for clients
- could use locks and sequencers with other systems

Boxwood

- 3 separate services
  - lock, Paxos, failure detection
  - could be used independently
- fewer, more sophisticated developers
- different default parameters
- lacks grace period
- uses locks primarily within

Summary

- Distributed lock service
  - course-grained synchronization for Google's distributed systems
- Design based on well-known ideas
  - distributed consensus, caching, notifications, file-system interface
- Primary internal name service
- Repository for files requiring high availability

References

The Chubby lock service for loosely-coupled distributed systems, Mike Burrows. Appears in Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI), November, 2006. http://labs.google.com/papers/chubby-osdi06.pdf

Paxos Made Simple, Leslie Lamport. Appears in ACM SIGACT News (Distributed Computing Column), Vol. 32, No. 4 (December 2001), pages 51-58. http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf Also,

Paxos Made Practical by David Mazieres http://www.cs.cornell.edu/courses/cs6464/2009sp/papers/paxos_practical.pdf