

CS349C
Chubby Lock Service
Mike Burrows, Google

Presented by:
Debangsu Sengupta
Michael Chan
2/2/10

Outline

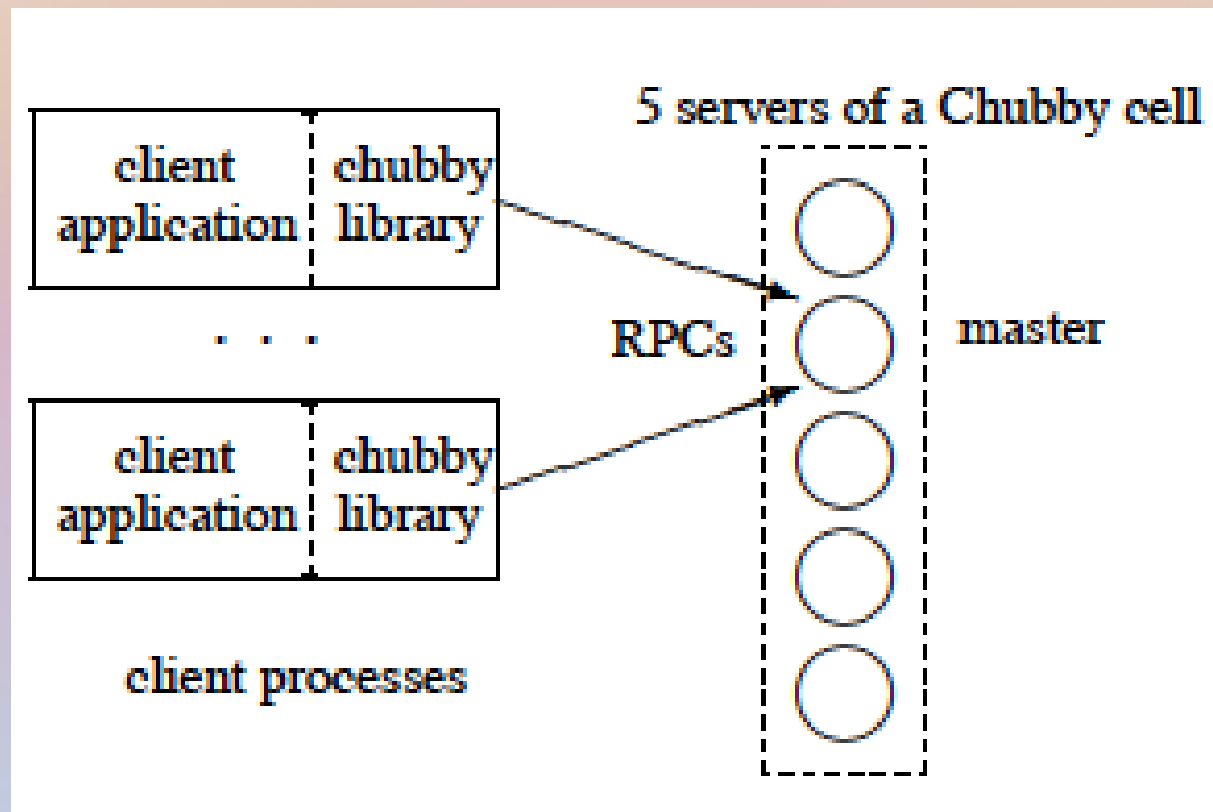
- Motivation
- Design in a nutshell
 - Example application
- Design details
- Scaling mechanisms
- Deployment experience

Motivation and Goals

- Many distributed systems at Google with ad-hoc primary election / human intervention
 - Distributed consensus problem
- System Features:
 - Coarse-grained locking
 - Small amounts of meta-data (e.g. primary's location)
 - Event notifications, e.g. master elected, master died, ...
- Automatic and more efficient primary election
- Goals:
 - Primary: Reliability, availability, clear + easy semantics
 - Secondary: throughput, storage capacity

Design in a Nutshell

- Lock API hiding Paxos (distributed consensus protocol) run by Chubby servers



GFS Example: Electing Master Server

- Clients: GFS servers
- All clients try to get lock X with exclusive mode on file `/ls/gfs/master`
 - `Acquire(X)`, `TryAcquire(X)`
- Client B wins lock, writes its IP address in file
 - `SetContents("1.2.3.4");`
- All other clients get notification(s)
 - e.g. “Lock X acquired” / File Modification event
- Clients A, C, D read `/ls/gfs/master` to get B's address
 - e.g. `GetContentsAndStat();`
- Where's Paxos? Chubby master runs it to perform leader elect. Exposed to clients as the lock acquirer.

Design Discussion

- Lock service
 - as opposed to distributed consensus library.
 - Simpler/familiar model, client needs 1 server to make progress.
- Storage for small files only (256 kB limit) – atomic r/w
- Use case: Readers/reads dominate
- Consistent caching of files
 - as opposed to polling
- Coarse grained locks:
 - apps responsible for fine-grained locks
 - Reduce load / availability requirements on lock server
 - Better survivability / less delay and stalls on lock server failure

Design: Chubby cell

- Set of replicas. Typical number: 5
- Master lease using Paxos. Renewed.
 - Promise to not elect new master till expiry
- Clients find master via replicas. Future comm with Master only.
- Simple db. Replicas maintain copy. Master issues r/w's.
- Writes: master → replicas: quorum using consensus.
- Reads: master handles queries (and proxies...).
- Replace failed replicas from periodic backups

Design Details

- Files, directories, handles
- Locks, sequencers
- Event notifications
- API
- Caching
- Sessions
- Failover
- Miscellaneous (DB implementation, backup, mirroring)

UNIX-like File System Interface

- FS interfaces support applications w/ specialized API's, GFS.
- No sym/hard links, path-independent ACL, no mv, no last-accessed/modified times
- Nodes: files / directories – permanent or ephemeral
- Metadata: names of 3 ACLs, 64-bit checksum, 64-bit monotonically increasing counters
 - Instance number: $>$ previous node w/ same name
 - content generation number
 - lock generation number
 - ACL generation number
- ACLs: ordinary files listing names of principals
 - R / W / change ACL name

File Handles

- Open() returns a handle like UNIX file descriptors
 - Check digits:
 - Prevent client guessing handles. Randomize.
 - Allows ACL checks only on open.
 - Sequence number, to distinguish between handles created by current / previous master
 - Mode information: Used by new master to recreate state for old handle.

Locks and Sequencers

- Any node can be a lock. Modes: Shared / exclusive (reader / writer)
- Advisory – can access file even if not holding lock, but generates “conflict” event. Write permission needed to acquire.
 - Allows reads to proceed for non lock-holder – admin/debug.
 - Chubby locks protect resources at remote services. Distributed locks adds system-wide complexity.
 - No value in mandatory locks and extra guards.
- Sequencers – byte strings (lock state) attached to ops involving locks to synchronize operations
 - Lock holder: `getSequencer()`. Remote Service: `checkSequencer()`
- Alt: Lock delay – If lock holder fails, other clients may not claim lock until after delay elapsed e.g. 1 min.
 - Prevents consistency issues if request from failed lock holder gets processed after lock is reacquired

Event Notifications

- Clients subscribe to asynchronous events upon handle creation.
- Eventing Model: Perform action, then event. Available on next read.
- Types:
 - File contents modified
 - Child node added, removed, or modified: (Mirror/ephemeral files)
 - Chubby master fail over: Event loss, rescan data.
 - Lock acquired
 - Conflicting lock request : allows lock caching
- Example:
 1. Lock (on file X) acquired → a primary is elected
 2. File X contents modified → primary location written, and available on next read by other clients

Chubby API

- Open()
 - Only call with node name. Access check always.
 - various parameters, e.g. handle mode, events, lock-delay
- Close(), Poison(), Delete()
- GetContentsAndStat() - File contents and metadata
- ReadDir() - like ls
- SetContents() - writes to a file atomically; SetACL()
 - Optional generation number param to set only if current.
- Acquire(), TryAcquire(), Release()
- Get / Set / CheckSequencer()
- All calls can be sync / async + get error / diagnostic info

Client-side Caching

- File data, node meta-data, absence of file
- Write-through memory cache, kept alive by leases
- Consistency enforced via invalidations from Chubby master.
 - Master writes block. Send invalidation events.
 - Call blocks till all caches return invalidation ACK + KeepAlive
 - Leverages # reads >>> # writes
 - No updates sent due to unbounded inefficiency.
- Strict consistency model easier for programmers.
- File handles – speed up multiple Open(). Subject to app restrictions.
- Locks – reuse locks, invalidated via “conflicting lock” event

Sessions

- Valid session → valid handles, locks, cache
- Idle session terminated after timeout
- KeepAlive RPC initiated by client to renew session; returned by master when
 - Current session ending soon, or
 - Invalidation needs to be performed (need ACK).
 - Blocking RPC model needed? Simpler?
- Lease extended on: session creation, Keep-Alive resp, Master failover.
- Client on local timeout
 - enter “jeopardy” state (disable cache). Event to app.
 - wait during “grace period” (45s).
 - find new master. Re-establishment via KeepAlive. Re-enable cache. “Safe” event to app. Else “expired” event.

Chubby Master Fail-over

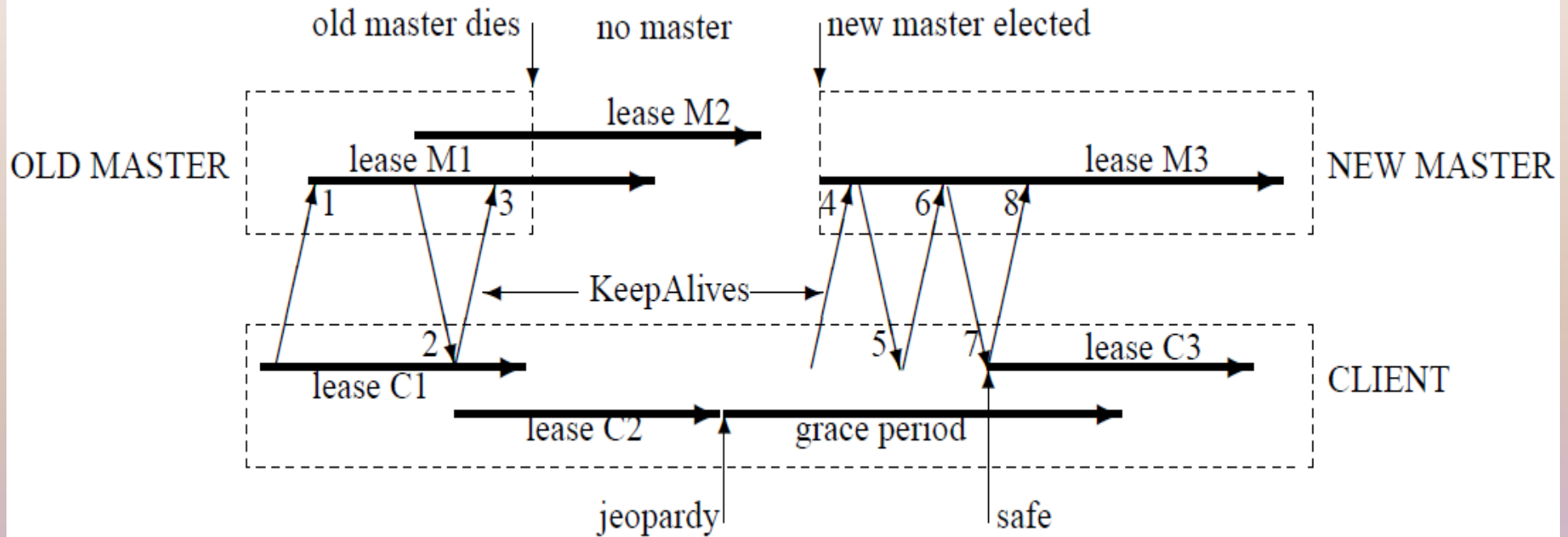


Figure 2: The role of the grace period in master fail-over

Master Fail-over

- Old master discards in-memory state: sessions, handles, locks.
- Extend lease / Stop timer till new master elected.
- Fast re-elect: client re-connect before lease expiry (before jeopardy).
- Slow re-elect: clients flush cache, enter grace period. Allow maintaining session across fail-over.
 - Blocks app calls to API till regaining session
- New master:
 - Selects new epoch num. Respond only to new master loc requests.
 - Reconstruct in-memory state from replicated DB (incl. ephem files)
 - Reply to client Keep-Alive. Send fail-over event.
 - Waits for all acks / let session expire. Allow new ops.
 - Old handle used: recreates in-mem handle representation. Allows.

DB Impl., Backup, Mirroring

- Database
 - Orig: Replicated Berkeley DB.
 - Switched to simpler DB w/ WAL, snapshots, atomic ops, but no transactions.
- Backup – Master writes DB snapshot to geographically distinct GFS server periodically
 - Disaster recovery.
 - New replica reduces / avoids load on in-service replica.
- Mirroring – copy files from a Chubby cell to another
 - Global cell mirrored to all other cells. /ls/global/master to /ls/cell/slave
 - Contains Chubby's ACLs, monitoring / entry points for clients of services, e.g. BigTable cells, configs.

Scaling mechanisms

- Biggest gain: Reduce communication with master
 - Arbit # of Chubby cells. Client pick nearby ones.
 - Increase lease time (12s to 60s). Reduce Keep-Alive frequency.
 - Client cache: file data, meta-data, handles.
- Proxies to handle KeepAlive and read requests
 - KeepAlive ~2K RPC/s. Cannot affect infrequent write traffic (< 1000 ppm)
 - Tradeoff: Add'l RPC for write/first time Read. Higher unavailability probability. Failover issues.
- Partitioning the namespace. Reduce R/W. No impact on KeepAlive.
 - Distribute load across multiple Chubby masters. Node D/C on $P(D/C) = \text{hash}(D) \bmod N$
 - Limit cross-partition traffic: ACL checks, dir. Deletion. Use cache.

Use and behavior

- Many naming related files
- Also common: config/
ACL, metadata files
- Avg: 10 clients per shared file.
- Few clients hold locks.
Shared locks uncommon.
- KeepAlive dominates RPC

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: Name Service

- DNS uses TTL values with expiry
 - MapReduce style shuffles caused huge loads with 3000 clients: 150K requests / s
- Better: consistent write through cache with push invalidations
 - Avoid polling. Optimize by returning names in batches.
- Protocol conversion servers:
 - serve time-based name resolution
 - DNS server for DNS clients e.g. browser

Abusive Clients

- Assume trusted environment. No DDOS.
- Manual reviews of Chubby usage
- Issues:
 - Lack of aggressive caching:
 - Poll on lock file, absent file, open file handles.
 - Lack of quotas:
 - 256kB quota – atomic r/w. Bigger files => diff. Store.
 - Publish/Subscribe
 - Not meant for general purpose pub/sub. => use XMPP

Fail-over problems

- Old design: Write session to DB on creation.
 - Overload when many procs started at once
 - => Store session on first modification/lock acq.
 - Spread out writes of active sessions across multiple Keep-Alives.
Small probability of loss for young r/o sessions. Problem: may read stale data on f/o.
- New design:
 - Sessions only in memory. Recreate like handles.
 - May wait till lease timeout to allow ops to proceed.
 - Support proxies:
 - Associate locks with new session (on a failover proxy).
 - Master needs to allow time for new proxy to grab locks/file handles.

Lessons learned

- Developers rarely consider availability
 - Best practice: Plan for short Chubby outages. Use coarse-grained lock.
 - Server Up != observed availability for a client.
- Fine-grained locking not needed
- Poor API choices: Repl. Close/Poison w/ Cancel for shared handles
- RPC use affects transport protocol:
 - TCP backoff within data center.
 - Keep-alive traffic over UDP. Augment with TCP GetEvent() in normal case

Related Work

Chubby

- 1 svc: Locks, reliable small-file store, session lease mech
- Wide target audience: experts/novice
- Higher level interface (combine lock and file name spaces)
- Clients cache by default
- Params: Default lease: 12s. K-A: 7s, 5 replicas
- Grace period
- Lost lock expensive for clients.
- Heavier weight locks + sequencers for external resources.

Boxwood

- 3 svcs: lock svc, Paxos (reliable state repository), failure detect svc. Can use indep.
- Narrower expert audience
- Clients likely to use Boxwood's caching
- Different default params e.g. 200ms ping. Timeout 1s. 2-3 replicas.
- No grace period
- Use lock within the system. Lighter-weight.

Summary

- Distributed lock service
 - Coarse grained activity sync for Google's D.S.
- Design based on well-known ideas
 - Distributed consensus, fault tolerance, consistent caching, event-notifications, fs interface
- Primary internal app-level naming system
- Repository for files with high availability reqs e.g. ACL's, config data.

References

- The Chubby Lock Service for Loosely-Coupled Distributed Systems, Mike Burrows, Google. OSDI'06.
- Chubby presentation, Xin (Joyce) Zhan. ([Link](#))
- Chubby presentation, Petro Nikolov ([Link](#))

END :)