THE CHUBBY LOCK SERVICE

For loosely-couple distributed systems! Written By: Mike Burrows

Presented By Oluwatosin V. Adewale

WHY?

- "To allow clients to synchronize their activities and agree on basic information about their environment…. We expected Chubby to help developers deal with coarse-grained synchronization within their systems, and in particular to deal with the problem of electing a leader..."
  - Mike Burrows

WHY?

- Nothing other than distributed consensus problem!

WHY?

- Aims to help developers via coarse-grained locking and with leader election, in particular.
  - However clients of Chubby can implement their fine-grained locks
- Seeks **reliability**, **availability**, **simple semantics** but not necessarily high performance
  - However, throughput and storage capacity not as important
- Engineering Effort not Scientific Research. No new techniques claimed.

CLIENT LIBRARY VS. LOCK SERVICE

- Library embodying Paxos or library accessing a centralized and reliable service?
- Lock service Pluses
  - Project start as prototypes not structured for use with a consensus protocol
  - However using a lock service for some consensus-dependent activity is not difficult with a reliable and available lock service.
  - Lock service gives mechanism for advertising result by storing small quantities of data as a file. Consistency features of protocol shared.
  - Single active clients can progress even when others are dead. They don't need to use quorums to make decision
    - Provision of a "consensus" service

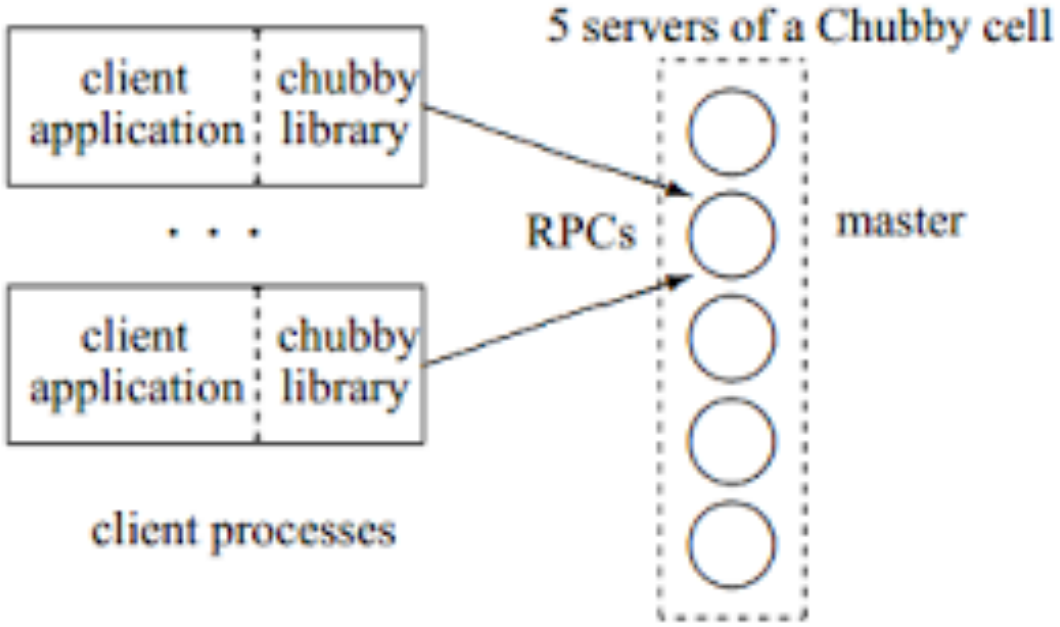
LOCK SERVICE FTW!!!

- Do the work or have someone do the work for you.
- Result. Chubby does the work:
  - Developers support their load
  - BUT do not worry about consensus
  - Also easy to integrate into their near-sighted designs / code vs. more complex consensus client library

DYNAMIC DUO

- Lock service for consensus
- Serve small-files for primaries to advertise themselves instead of using a second service, as advertising oneself as the rightful leader is challenging in itself.

HOW? (SYSTEM ARCHITECTURE)



HOW? (SYSTEM ARCHITECTURE)

- Chubby Client Library
  - Links to client application
  - Mediator between client and server
  - Clients find the master by reaching out to replicas listed in the DNS (domain name service)
  - Send request to master until it ceases to respond or says it is no longer the master.

HOW? (SYSTEM ARCHITECTURE)

- Chubby cell (servers)
  - Where consensus actually happens
  - 5 replicas. Maintain copies of a simple database.
  - Master elected with majority of votes via consensus
  - Each voter promises not to elect a different master for some seconds
    - known as a ***master lease***
  - Master alone writes and reads from DB. Reads safe if master lease not expired.
  - Replicas copy updates from master via consensus protocol
  - Replicas respond to client request with identity of master.
  - Replicas acknowledge master's write updates
  - Master fails, new election run.
  - Replica fails for few hours, replaced from free pool. DNS, Replicas, and new replica updated
  - Most chubby cells confined to some room. Global cells has servers far apart.

HOW?

- Consistent client Caching and not Time-based caching
  - Strength as a name service vs. DNS
  - Poorly chosen times can lead to high DNS load, people keep asking for IP address of the same server or long client fail-over times, have to wait for longer before we know that ip address has changed
  - Whole file reads and writes, supports whole file caching.
  - Do not deal with issues of concurrent slight modification.
  - Just invalidate file and re-cache
- Use of leases
  - Master lease (few seconds) to make sure reads are safe and new leader elected if master fails
  - Periodically renewed by replicas provided master continues to win a majority of votes
  - Session leases
- Available with its own specialized API and via interfaces used by their other file systems such as GFS.

MORE ON CACHING

- Reduces traffic by caching file data and node meta-data.
- Consistent Write-through Cache
- Maintained by lease mechanism.
  - You either have a consistent cache or not, error
- When data is changed, master sends invalidations and modification blocked until
  - All affected clients flush cache and acknowledge or lease expires
- Caches are only invalidated, never updated
  - Prevents unnecessary updates sent to client
- Reads processed without delay
- Can cache open handles and locks

CHUBBY FILE SYSTEM INTERFACE

- Unix-like file system. Files, directories and handles.
- ls/fu/bark/the/rest
  - ls – lock service
  - fu – name of chubby cell, could be *local* or *global*
  - The remainder of the name is interpreted within the named Chubby cell.
- Client receives handler when files and directories (nodes) opened.
  - Similar to Unix File Descriptors
  - Contains extra meta-data to prevent clients creating or guessing handles, for current master to distinguish handlers created by previous master
  - Information for new master to recreate handler's state

CHUBBY FILE SYSTEM INTERFACE

- Different from Unix file system in different ways:
  - No path-dependent permission semantics. Permission per file
  - Operations to move file from one folder to the other not exposed
  - Files and directories collectively called nodes. Can be permanent or ephemeral.
  - Ephemeral nodes can be deleted explicitly but deleted if not opened by a client or if node directory is empty (kind of like garbage collection)
  - Nodes have meta data including ACLs, instance number, content generation number, a lock generation number and 64-bit file content checksum to check if files differ.

HOW?

- Locks
  - Each Chubby file and directory can act as a reader-writer lock
  - Locks are advisory.
  - Lock-delay. If lock holder fails or becomes inaccessible, lock cannot be acquired for bounded time period specified by locker (one minute at most)
- Use of sequencers
  - Simplifies locking in distributed system
  - Allows clients to verify another client's ownership of lock. E.g. file server verifying that client does indeed have lock on a file

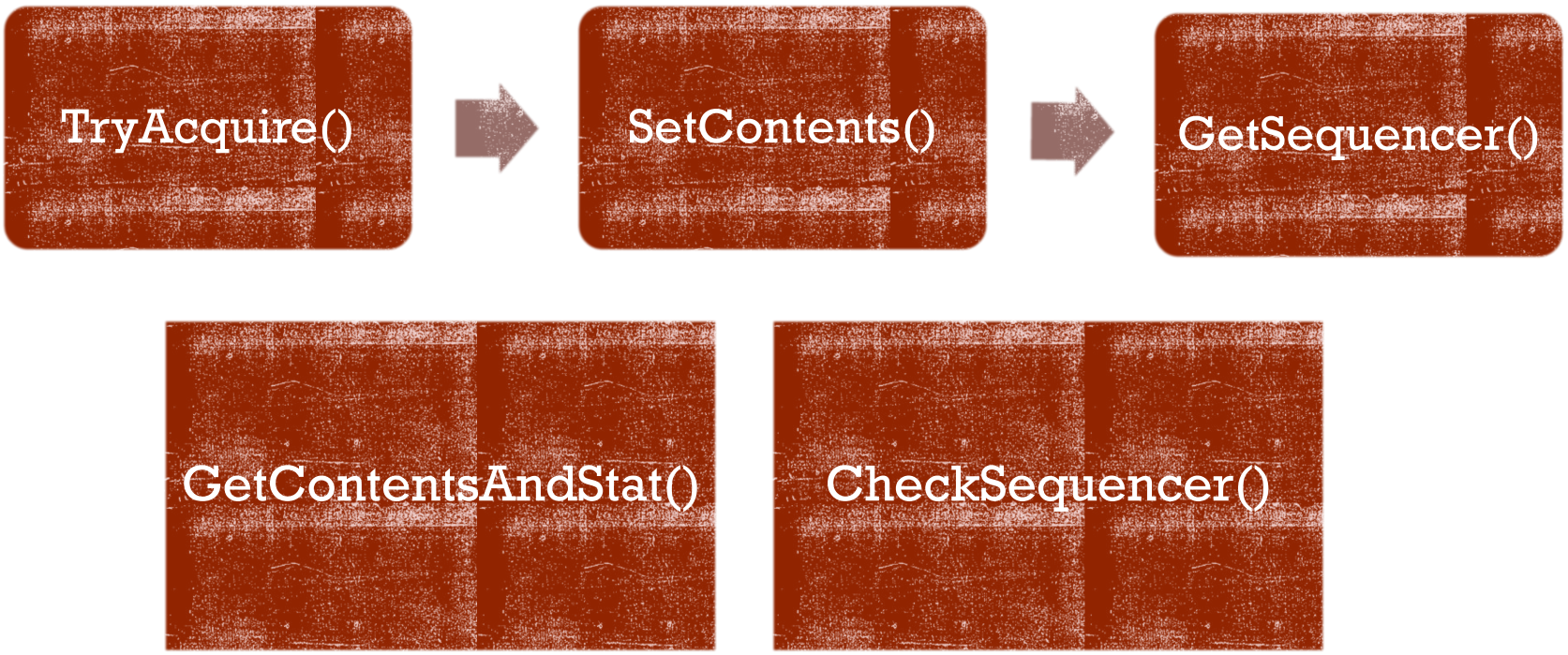
HOW?

- Events
- Clients can subscribe to events when they create a handle. Delivered to clients asynchronously
- Include:
  - File contents modified
  - Child node added, removed or modified
  - Chubby master failed over.
  - Handle and its lock has become invalid
  - Lock acquired
  - etc

HOW (API)

- Operations can be carried on handle that points to some opaque structure
- API:
  - Open()
    - Opens named file with various options. Succeeds if client has correct permissions
  - GetContentAndStats()
  - SetContents()
  - Delete()
  - Acquire(), TryAcquire() and Release() for locks.
  - GetSequencer(), SetSequencer(), and CheckSequencer()
- All calls take an operation parameter in addition
  - Allows callbacks, waiting for completion of call obtain more information

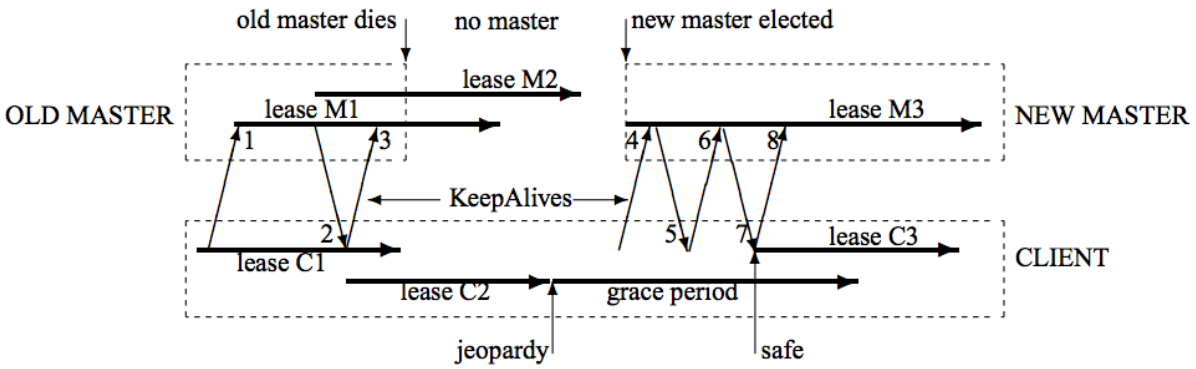
PRIMARY ELECTION USING THE API



SESSIONS AND KEEPALIVES

- Session. Relationship between Chubby Client and Cell
- Sustained through handshakes called KeepAlives
  - Keepalives also pass events and cache invalidation to client.
- Cache remains valid provided the session remains valid.
- Session ends when terminated or its been idle
- Every session has a lease timeout on both master and client. Client more conservative on estimate of duration of lease
  - Session guaranteed not to be terminated by master during lease
  - Master can advance (or pause) it but not reverse it in time for special situations.
- As session is maintained master extends lease at least every 12s. Higher with more loads.

FAILOVERS



FAILOVERS

- If election takes too long
- Client in grace period
- Flush and try to find master in grace period
- Allows session to be maintained across failovers
- Client in jeopardy and lets app know so it can quiesce itself until it can be sure of status of session
- If grace period exceeded, session abandoned and report failure to application
- Else new master and client lib co-operate to give illusion that no failure has occurred.
- Master conservstively approximates in-memory state of prev master
- Reads data that was replicated on disc

TIDBITS

- Uses features of Berkeley DB for database log distributed among replicas
- Every few hours writes snapshot of database to a GFS server in diff building
  - Allows backup to survive whole site failure
  - Prevents cyclic dependencies.
  - Allows disaster recovery and way ti initialize db of new replica without burdening in-use replicas
- Allows mirroring of files (often config) to clusters distributed world-wide from one cell to another
- Fast due to small data and event mechanism
- Global cell mirrors some data in each Chubby cell. Allows worl-wide accessibility.

SCALE

SCALE

- Chubby can also deal with moderately large scale (90,000 clients comm. with one master)
- Supports moderately large scale:
  - Use more cells
  - Increase lease times (12s to 60s)
  - Caching largely reduces function calls (not necessarily keepalives rpcs)
- Protocol-conversion servers
  - Proxies
  - 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

RESULTS

- Many files used for naming
- Fail-overs relatively infrequent but take a bit when they occur
- Most locks are exclusive and few hold them.
- Approx 10 clients use each file (230k / 24k)
- RPC is dominated by sesaon keep Alives,
- Few reads, fewer writes and even fewer lock acquisitions
- Data loss been primarily due to database error and operator error. Implies chubby is very reliable.

MOAR INFO...

MOAR RESULTS AND OBSERVATIONS

- Operations typically have low latency.
- 1ms for local read, 10ms for writes due to locking.
  - Write latency can be increased further due to recently-failed client with cached file
- Dealing with Java clients a bit complicated. Solution was to use a protocol-conversion server.
- Use as a name service despite being designed as a lock service!!!
  - TTL can be difficult to choose and isn't responsive. If ttl too low, dns servers overloaded
  - If too high servers not replaced promptly, still large number of lookups for many clients
  - Explicit invalidation means that keepalives maintain abitrary number of entries

EVEN MOAR...

- Abusive clients. Not intentional or malicious.
- Often due to mistakes, misunderstanding and differing expectations.
- Lack of aggressive caching leading to unnecessary function calls
- Lack of quotas for storage. More services started using a lot of data, which could slow speed of transactions. They introduced 256 kB limit on file size.
- Slow and inefficient for most publish/subscribe situation due to heavyweight guarantees and use of invalidation instead of updates to maintain cache consistency
- Poor developer practices, like not considering availability in their design. Treated chubby like it was always available. So long recovery procedures during failovers.
- Poor api choices, like no cancel() but only close() or posion()
- RPC use affects transport protocol. When TCP loaded, backs odd leading to lost sessions. So they used UDP for keep alive RPCs

FINALLY . . .

- Reactive design choices were made as Chubby was used
- Chubby was used in different way designers envisioned
- Centralization of consensus has benefits but also costs and constraints
- Delays can be okay in distributed system and make consensus easier, as long as they are bounded by an agreed / pre-determined amount. Stronger guarantees but lower performance
- Strong consensus guarantees can be difficult!!!
  - Especially if you want low latency and simplicity
- Simplicity can reduce flexibility
  - Whole file reading and cache invalidation without updates
- Even simple things can be hard to understand (even at google)
  - It is important to factor client (programmer) behavior in design or at least be aware of need to educate them

CONSENSUS

- Consensus is difficult.
- This relatively complex system was designed so that nodes can agree on values and information.

USES

- Google File System for electing master
- Bigtable (compressed, high performance data storage system):
  - Leader election
  - Membership testing
  - Allow clients to find master (advertisements)
- Meta data storage for small amounts of data. Use as root of datastructure
- Partition work at a coarse-grain between servers
- As a name service

SOURCES

- Chubby Lock Service for loosely-couple distributed systems: https://static.googleusercontent.com/media/research.google.com/en//archive/chubby-osdi06.pdf
- Baby Images: Google

THANK YOU!!!