CSE 487/587
Data Intensive Computing

Lecture 14: Chubby
Part 2 of noSQL

Vipin Chaudhary

vipin@buffalo.edu

716.645.4740
305 Davis Hall

Overview of Lecture Series

- Chubby
- BigTable
- Spanner
- Zookeeper
- Dynamo

The Chubby lock service for loosely- coupled distributed systems

Slides from Mike Burrows @ Google, OSDI 2006

An Engineering Effort

- 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 (as opposed to high performance)
- Use?
  - Used in Google: GFS, Bigtable, etc.
  - Elect leaders, store small amount of meta-data, as the root of the distributed data structures

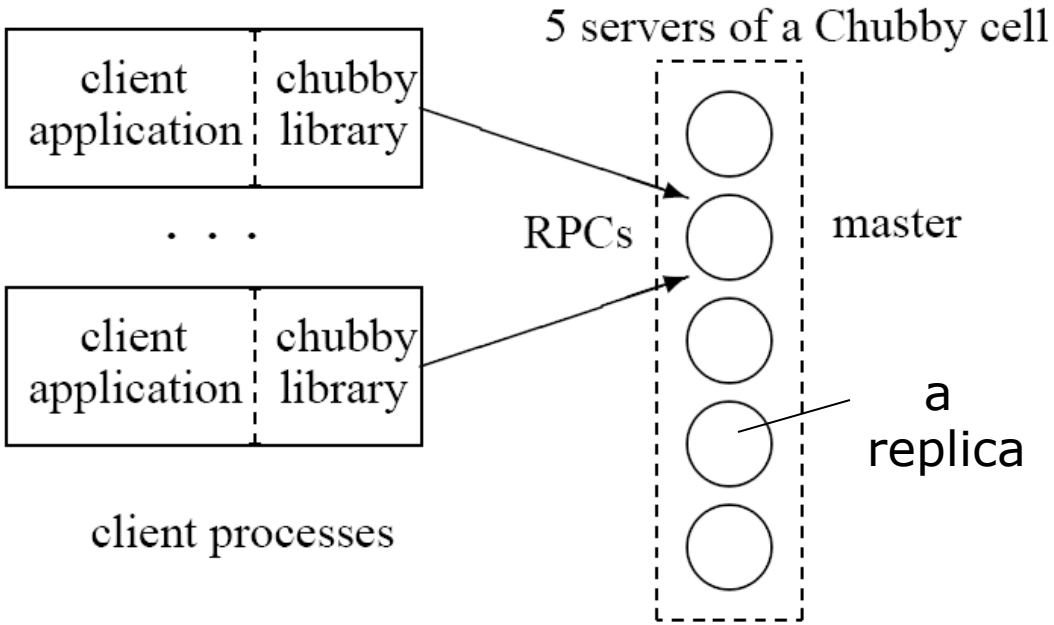
More to Come

- Design
- Mechanisms for scaling
- Use, surprises and design errors
- Summary

Design Decision

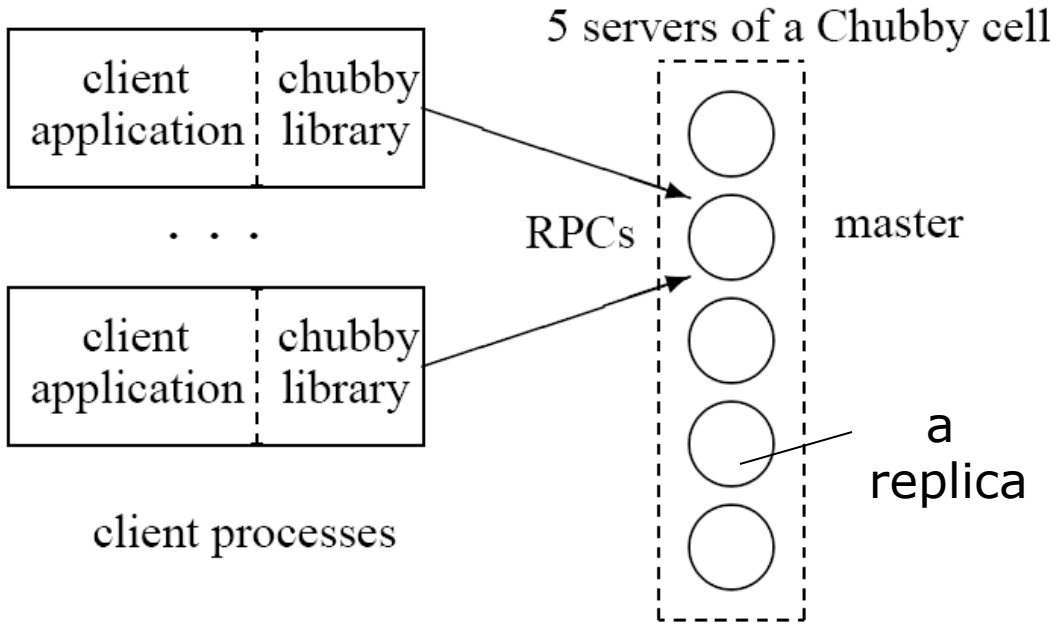
- Serve small-files to permit elected primaries to advertise themselves and their parameters, rather than build and maintain a second service
  - A service advertising its primary via a Chubby file may have thousands of clients.
    - must allow thousands of clients to observe this file, preferably without needing many servers.
  - Clients and replicas of a replicated service may wish to know when the service's primary changes
    - event notification mechanism would be useful to avoid polling
  - Even if clients need not poll files periodically, many will; this is a consequence of supporting many developers.
    - caching of files is desirable
    - Consistent caching is desirable
  - Provide security mechanism like access control
  - Do not expect lock use to be fine grained (seconds)
    - Rather it is hours and days.

System Structure



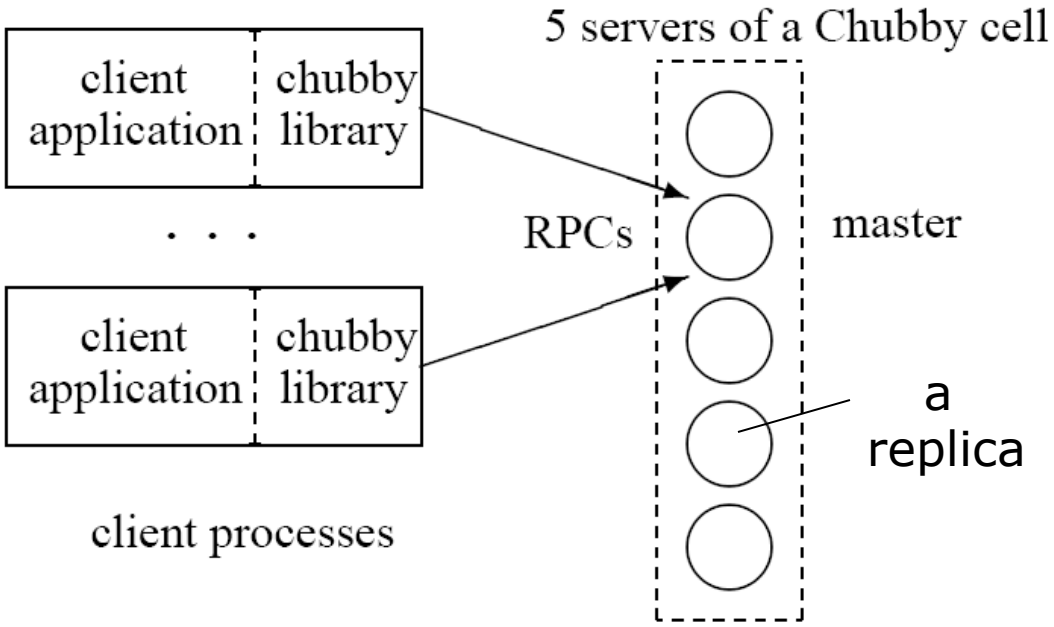
- A chubby cell consists of a small set of servers (replicas)
- A master is elected from the replicas via a consensus protocol
  - Master lease: several seconds
  - If a master fails, a new one will be elected when the master leases expire
- Client talks to the master via chubby library
  - All replicas are listed in DNS; clients discover the master by talking to any replica

System Structure (2)



- Replicas maintain copies of a simple database
- Clients send read/write requests only to the master
- For a write:
  - The master propagates it to replicas via the consensus protocol
  - Replies after the write reaches a majority of replicas
- For a read:
  - The master satisfies the read alone

System Structure (3)



- If a replica fails and does not recover for a long time (a few hours)
  - A fresh machine is selected to be a new replica, replacing the failed one
  - It updates the DNS
  - Obtains a recent copy of the database
  - The current master polls DNS periodically to discover new replicas

UNIX-like File System Interface

- Chubby supports a strict tree of files and directories
  - No symbolic links, no hard links
  - /ls/foo/wombat/pouch
    - 1st component (ls): lock service (common to all names)
    - 2nd component (foo): the chubby cell (used in DNS lookup to find the cell master)
    - The rest: name inside the cell
  - Can be accessed via Chubby's specialized API / other file system interface (e.g., GFS)
- Support most normal operations (create, delete, open, write, …)
- Support advisory reader/writer lock on a node

ACLs and Metadata

- Access Control List (ACL)
  - A node has three ACL names (read/write/change ACL names)
  - An ACL name is a name to a file in the ACL directory
  - The file lists the authorized users
- Per node metadata includes 4 monotonically increasing 64-bit numbers that allow clients to detect changes
  - an instance number; greater than the instance number of any previous node with the same name.
  - a content generation number (files only); this increases when the file's contents are written.
  - a lock generation number; this increases when the node's lock transitions from free to held .
  - an ACL generation number; this increases when the node's ACL names are written.
- Chubby also exposes a 64-bit file-content checksum so clients may tell whether files differ

File Handles

- File handle: (analogous to Unix file descriptors)
  - Has check digits encoded in it; cannot be forged
    - Check digits prevent clients from creating or guessing handles
    - Full access control checks need be performed only when handle created
      - Unlike Unix at each read/write
  - Sequence number:
    - a master can tell if this handle is created by a previous master
  - Mode information at open time:
    - If previous master created the handle, a newly restarted master can learn the mode information (recreate its state)

Locks and Sequences

- Locks: advisory rather than mandatory
- 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 (typically 1 min)
- Solution #2: sequencer
  - A lock holder can obtain a sequencer from Chubby
    - an opaque byte-string that describes the state of the lock immediately after acquisition.
    - contains the name of the lock, the mode in which it was acquired (exclusive or shared), and the lock generation number.
  - It attaches the sequencer to any requests that it sends to other servers (e.g., Bigtable)
  - The other servers can verify the sequencer information

Chubby Events

- Clients can subscribe to events (up-calls from Chubby library) when they create a handle
  - Events delivered to clients asynchronously
- Events include
  - File contents modified: if the file contains the location of a service, this event can be used to monitor the service location
  - Master failed over
    - Warns clients that other events may have been lost, so data must be rescanned
  - Child node added, removed, modified (used for mirroring)
  - Handle becomes invalid: probably communication problem
  - Lock acquired (rarely used)
    - Can be used to determine when a primary has been elected
  - Locks are conflicting (rarely used)
    - Conflicting lock request from another client – allows the caching of locks

APIs

- Open()
  - Opens a named file or directory to produce a handle
  - Open takes a node name; all others operate on handles
  - Mode: read/write/change ACL; Events; Lock-delay
  - Create new file or directory?
- Close() – this call never fails
- Poison()
  - causes outstanding and subsequent operations on the handle to fail without closing it;
  - allows a client to cancel Chubby calls made by other threads without fear of deallocating the memory being accessed by them
- GetContentsAndStat()
  - returns both the contents and meta-data of a file.
  - contents of a file are read atomically and in their entirety.
  - GetStat() – just metadata
  - ReadDir() – names and metadata for children of directory

APIs

- SetContents(): set all contents;
  - The contents of a file are always written atomically and in their entirety
  - Optionally, the client may provide a content generation number to allow the client to simulate compare-and-swap on a file; the contents are changed only if the generation number is current..
  - SetACL() – similar to SetContents on the ACL names associated with the node
- Delete() – deletes node if it has no children
- Locks: Acquire(), Release()
- Sequencers: GetSequencer()
  - Returns a sequencer that describes any lock held by this handle.
- SetSequencer()
  - associates a sequencer with a handle. Subsequent operations on the handle fail if the sequencer is no longer valid.
- CheckSequencer() – checks if sequencer is valid

Example: Primary Election

- All potential primaries open the lock file and attempt to acquire the lock.
- One succeeds and becomes the primary, while the others act as replicas.

```
Open("write mode");
If (successful) {
      // primary
   SetContents("identity");
}
Else {
   // replica
   open ("read mode", "file-modification event");
   when notified of file modification:
            primary= GetContentsAndStat();
}
```

- Primary obtains a sequencer with GetSequencer() ,which it then passes to servers it communicates with
- they should confirm with CheckSequencer() that it is still the primary.

Caching

- Strict consistency: easy to understand
  - Lease based
  - master will invalidate cached copies upon a write request
- Write-through caches

Sessions, Keep-Alives, Master Fail-overs

- Session:
  - 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 (but may increase time if master busy)
- Clients maintain a local timer for estimating the session timeouts (time is not perfectly synchronized)
- If local timer runs out,
  - Unsure whether master has terminated session
  - Client empties and disables its cache – session in *jeopardy*
  - Client waits for a 45s grace period before ending the session
  - When session survives, *safe* event tells client to proceed
  - If session timed out, *expired* event sent out
  - Happens when a master fails over

Explanation of fail-overs

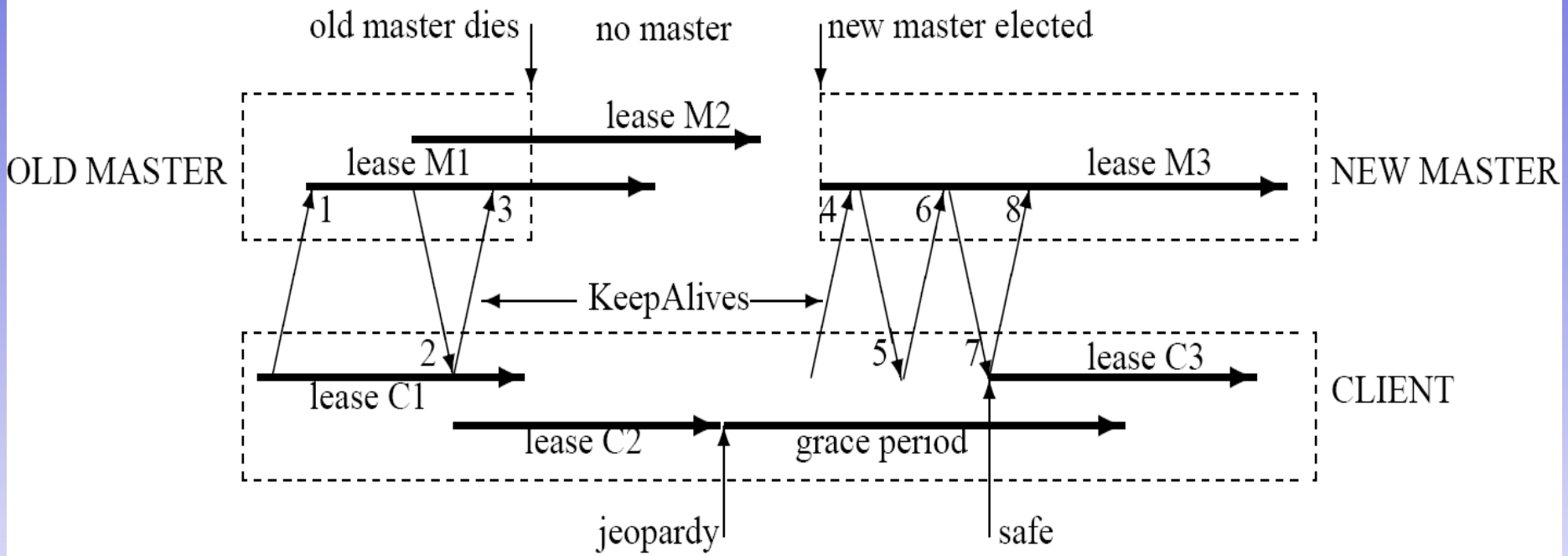


Figure 2: The role of the grace period in master fail-over

Other Details

- Database implementation
  - a simple database with write ahead logging and snapshotting
  - Database log is distributed among the replicas using distributed consensus algos
- Backup (every few hours):
  - 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.)
  - Fast as files are small

More to Come

- Design
- Mechanisms for scaling
- Use, surprises and design errors
- Summary

Why scaling is important?

- Clients connect to a single instance of master in a cell
  - Much more client processes that number of machines (90K clients for a master)
  - So reduce communication with master
- Existing mechanisms:
  - More Chubby cells (typically one chubby cell for DC of 1000s machines)
  - Increase lease time from 12s to 60s to reduce KeepAlive messages (dominant requests in experiments)
  - Client caches to reduce calls to server

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

Almost the end

- Design
- Mechanisms for scaling
- Use, surprises and design errors
- Summary

Typical cell

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

in a 10-minute period

Cell outages Stats

- 61 outages for all cells in a few weeks (700 cell days)
  - most outages were 15s or less
  - 52 outages were under 30s
  - incur under 30s delay in applications
  - The remaining nine outages were caused by network maintenance (4), suspected network connectivity problems (2), software errors (2), and overload (1).
- In a few dozen cell-years of operation, we have lost data on six occasions, due to database software errors (4) and operator error (2); none involved hardware error.

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

Use 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

Chubby Summary

- Lock Service
- Loosely-couple distributed system
- UNIX-like file system interface
- Reliability and availability