The Chubby Lock Service

The Chubby Lock Service is a distributed course-grained locking service. A lock service is a form of consensus service that converts the problem of reaching consensus to handing out locks. Basically, a distributed set of processes compete to acquire a lock. The process with the lock has the baton to act on a target resource. Apache ZooKeeper is the open-source alternative to Chubby.

Chubby also provides low-volume storage to be used as a repository for distributed systems' configuration changes. Chubby's most popular use has been as a name service.

Overview

Chubby's goal is to allow clients to synchronize their activities and agree to basic information about their systems. Clients need to ensure that only one process across their entire system will act on a resource. Examples include:

- Ensure only one server can write to a database or write to a file.
- Ensure only one server can perform a particular action.
- Ensure there is a single master that processes all writes.

Chubby uses the Paxos distributed consensus protocol to solve for asynchronous consensus. Each Chubby instance (cell) is dedicated to a single data center and typically serves about 10,000 machines.

These notes do not go into the details of Paxos distributed consensus. To learn about Paxos, read Paxos Made Simple by Leslie Lamport. Also, Raft is very similar to Paxos. Refer to the notes on Raft Distributed Consensus.

Design

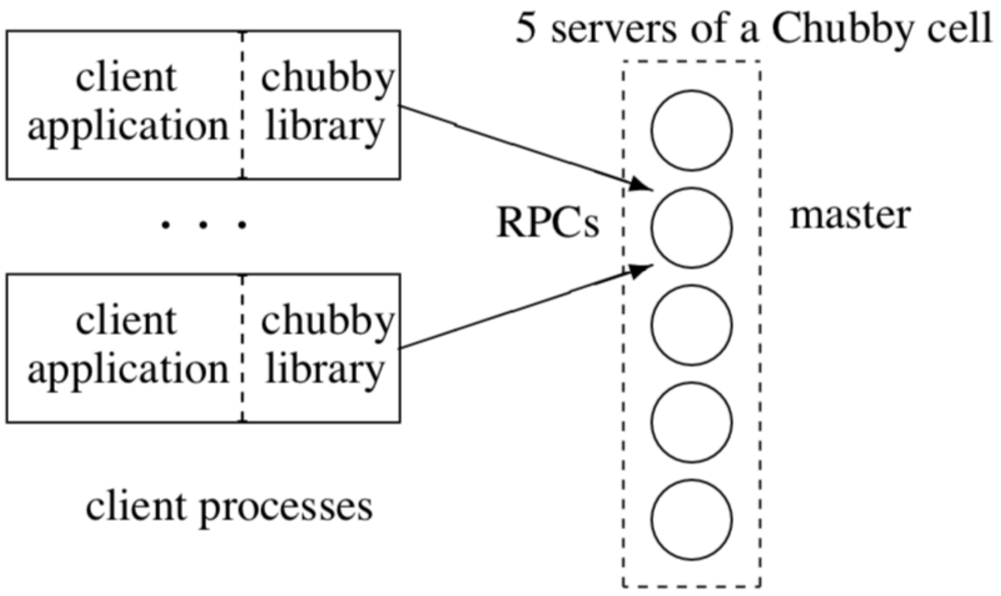


Figure 1: System structure

Chubby wasn't designed to be a library that only provides Paxos distributed consensus, but instead as library that accesses a centralized lock service. This makes it easier for developers to maintain their existing program structures. Minimal effort is required to elect a master and write to an existing file server.

The storage feature is important as services need to advertise Chubby's results with others e.g., after a primary is elected or when data is repartitioned. Not having a separate service for sharing the results reduces the number of servers that clients depend on. Another benefit is that the consistency features of the protocol become shared.

Each Chubby cell typically has 5 replicas. At the same time, read requests are handled by the master only. A service advertising its primary through a Chubby file can have thousands of clients that need to observe this file. 93% of RPCs are KeepAlives between a Chubby client and a Chubby cell.

Chubby uses event notification to notify clients of any changes to Chubby files. For example, clients and replicas of a service will need to be notified if the master/primary changes. Most common events include file contents modified, child node added/removed/modified, and Chubby master failed over.

Coarse-Grained Locks versus Fine-Grained Locks

Chubby utilizes coarse-grained locks (hours or days) over fine-grained locks (seconds or less). Coarse-grained locks are less load-intensive on the lock service. Lockacquisition rates are rare in comparison to a client's transaction rates. Clients are not significantly delayed by the temporary unavailability of a lock server.

With fine-grained locks, even a brief unavailability of a lock server would cause many clients to stall. If client require fine-grained locks, it is straightforward for clients to implement their own into their applications.

Locks and Sequencers

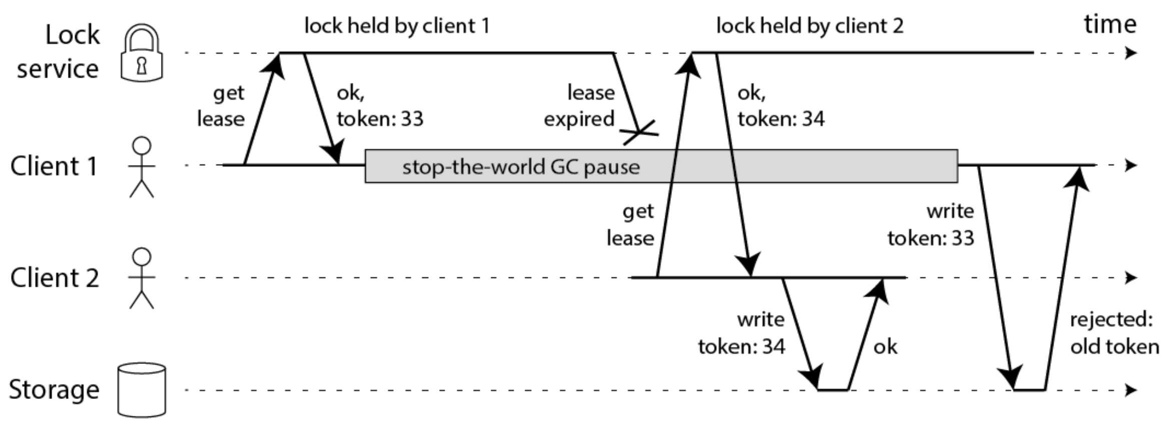
For writes, only one client can hold the lock in exclusive (writer) mode. For reads, any number of clients can hold the lock in shared (reader) mode.

Chubby uses advisory (cooperative) locks over mandatory locks--the lock doesn't prevent other clients from accessing a file. With advisory locks, clients are expected to check for a lock prior to taking action on a file. Google has standards to ensure developers perform error checking by writing `assert(lock X is held)`. This minimizes the benefit provided by mandatory locks.
Mandatory locks prevent users from accessing a locked file for debugging or administrative purposes. If a file must be accessed, an entire application would need to be shutdown or rebooted to break the mandatory lock. Mandatory locks require extensive modifications and overhead to Chubby to support them in a meaningful way.

With distributed systems, receiving messages out of order is a problem that needs to be addressed--Chubby uses sequence numbers to do so. A client can request for a sequencer on files it holds a lock on. A sequencer is a byte-string that contains:

- Name of the lock
- Lock mode (exclusive or shared)
- Lock generation number

The client passes this sequencer to the appropriate file servers. The file servers are expected to validate the sequencer and protect the client's operations in the given lock mode.



For file servers that do not support sequencers, Chubby provides a lock-delay period--typically one minute. The lock-delay protects from regular problems caused by message delays and restarts.

Caching

To reduce read traffic to the Chubby cell, Chubby clients cache metadata and file data in a write-through cache on the client's memory. The following occurs to keep cache consistent:

1. The master maintains a list of each client's cache contents.
2. When metadata or file data is to be changed, the master blocks the modification.
3. The master sends invalidations on top of the KeepAlive RPCs to every client that has cached this data.
4. When a client receives the invalidation, it flushes the invalidated data from cache and acknowledges the change to the master in its next KeepAlive call.
5. Once all clients acknowledge the invalidation, the master proceeds with the modification.

In Chubby, caching is important as read requests greatly outnumber write requests. Also, invalidation protocols are more efficient than update protocols. Update protocols require clients that have accessed a file once to receive updates for that file indefinitely--this is unnecessary.

Failover

When a Chubby master fails, it loses its in-memory state on sessions and locks. When a new master is elected, the following occurs:

1. The master builds in-memory data structures for sessions and locks that are recorded in the database.
2. Clients resume KeepAlives to the master, but no other operations.
3. The master sends a failover event for each client to flush their caches.
4. Once a client acknowledges the failover event, the master allows all operations from that client to proceed.

Example: Chubby Name Service

Chubby's success as a name service is due to its use of consistent client caching (distributed consensus) over time-based caching. Developers appreciate not having to manage a cache timeout such as DNS's time-to-live (TTL) value.

Caching within the standard internet naming system, DNS, is based on time. Each DNS entry has a TTL.

- Long TTL values lead to long client fail-over times.
- Short TTL values will overload DNS servers.

Chubby's caching uses explicit invalidations--a constant rate of session KeepAlive requests can maintain a large number of cache entries indefinitely.

A single Chubby name service cell is able to handle 90,000 clients communicating directly with it.

Examples

- **Google File System**. Chubby is used to appoint a GFS master server. It is also used to store metadata. Chubby is the root of its distributed data structures.
- **Bigtable**. Chubby is used to elect a master, allow the master to discover the servers it controls, and allow clients to find the master. It is also used to store metadata. Chubby is the root of its distributed data structures.
- **Access Control Lists (ACLs).** Chubby is used as a repository for ACL files.