Goel, Ajay: Nu Id (001897443)

**INFO 7250**
**Engineering Big-Data Systems**
**Summer Full 2019**
**Assignment 1**

# The Chubby lock service for loosely-coupled distributed systems

Chubby provides an interface much like a distributed file system with advisory locks, but the design emphasis is on availability and reliability, as opposed to high performance.

The purpose of the lock service is to allow its clients to synchronize their activities and to agree on basic information
about their environment. The primary goals included reliability, availability to a moderately large set of clients, and easy-to-understand semantics.

- Deal with coarse-grained synchronization within their systems, and in particular to deal with the problem of electing a leader from among a set of otherwise equivalent servers.

**How chubby is used?**

- To elect a master, to allow the master to discover the servers it controls, and to permit clients to find the master.
- Bigtable use Chubby as a well-known and available location to store a small amount of meta-data; in effect they use Chubby as the root of their distributed data structures. Also, to partition work between several servers.

**Design:**
Lock Service Advantages:

- Sometimes developer do not plan for high availability in the way one would wish. Often their systems start as prototypes with little load and loose availability guarantees. Invariably the code has not been specially structured for use with a consensus protocol. As the service matures and gains client's availability becomes more important.
- Allow clients to store and fetch small quantities of data, to read and write small files. This could be done with a name service, but it has been that the lock service itself is well-suited for this task, both because this reduces the number of servers on which a client depends and because the consistency features of the protocol are shared.

Some decisions follow from expected use:
• A service advertising its primary may have thousands of clients. Therefore, we must allow thousands of clients to observe this file.
• Clients and replicas of a replicated service may wish to know when the service's primary changes. This suggests that an event notification mechanism would be useful to avoid polling.
• Even if clients need not poll files periodically; this is a consequence of supporting many developers.
• Developers are confused by non-intuitive caching semantics, prefer consistent caching.
• To avoid both financial loss and jail time, it provides security mechanisms, including access control.

Coarse-grained locks impose far less load on the lock server. The transfer of a lock from client to client may require costly recovery procedures, so one would not wish a fail-over of a lock server to cause locks to be lost. Performance and the ability to add new servers at will are of great concern because the transaction rate at the lock service grows with the combined transaction rate of clients.
Benefits are that client developers become responsible for the provisioning of the servers needed to support their load and are relieved of the complexity of implementing consensus themselves.

**System Architecture:**
Chubby has 2 components: a server and a library that client applications link against.

- Chubby cell consists of a small set of servers known as replica. The replicas use a distributed consensus protocol to elect a master; the master must obtain votes from a majority of the replicas, plus promises that those replicas will not elect a different master for an interval of a few seconds known as the master lease. The master lease is periodically renewed by the replicas provided the master continues to win a majority of the vote.
- Clients find the master by sending master location.
- Once a client has located the master, the client directs all requests to it either until it ceases to respond, or until it indicates that it is no longer the master.
- If a replica fails and does not recover for a few hours, a simple replacement system selects a fresh machine from a free pool and starts the lock server binary on it.
- It then updates the DNS tables, replacing the IP address of the failed replica with that of the new one. The current master polls the DNS periodically and eventually notices the change.
- It then updates the list of the cell's members in the cell's database.

**Locks and Sequencers:**
Each Chubby file and directory can act as a reader-writer lock.

- Chubby locks often protect resources implemented by other services, rather than just the file associated with the lock.
- In a complex system, it is harder to use the approach employed on most personal computers, where administrative software can break mandatory locks simply by instructing the user to shut down his applications or to reboot.

**Events:**
Chubby clients may subscribe to a range of events when they create a handle.
Events include:

- file contents modified often used to monitor the location of a service advertised via the file.
- Child node added, removed, or modified used to implement mirroring
- Chubby master failed over warns clients that other events may have been lost, so data must be rescanned.
- Handle has become invalid; this typically suggests a communications problem.
- Lock acquired can be used to determine when a primary has been elected.
- Conflicting lock request from another client allows the caching of locks.
- Events are delivered after the corresponding action has taken place.

**Caching:**

- To reduce read traffic, Chubby clients cache file data and node meta-data in a consistent, write-through cache held in memory. The cache is maintained by a lease mechanism described below and kept consistent by invalidations sent by the master, which keeps a list of what each client may be caching.
- The protocol ensures that clients see either a consistent view of Chubby state or an error.
- Only one round of invalidations is needed because the master treats the node as uncacheable while cache invalidations remain unacknowledged.
- The caching protocol is simple: it invalidates cached data on a change, and never updates it.
- Caching is restricted in minor ways so that it never affects the semantics observed by the client: handles on ephemeral files cannot be held open if the application has closed them and handles that permit locking can be reused but cannot be used concurrently by multiple application handles.

**Sessions and KeepAlives:**
A Chubby session is a relationship between a Chubby cell and a Chubby client; it exists for some interval of time and is maintained by periodic handshakes called KeepAlives.

The master advances the lease timeout in three circumstances:

- On creation of the session, when a master fail-over occurs, and when it responds to a KeepAlive RPC from the client. On receiving a KeepAlive, the master typically blocks the RPC until the

client's previous lease interval is close to expiring. The master later allows the RPC to return to the client, and thus informs the client of the new lease timeout.

- The master may extend the timeout by any amount. The default extension is 12s, but an overloaded master may use higher values to reduce the number of KeepAlive calls it must process. The client initiates a new KeepAlive immediately after receiving the previous reply. Thus, the client ensures that there is almost always a KeepAlive call blocked at the master.

**Fail overs:**

- When a master fails or otherwise loses mastership, it discards its in memory state about sessions, handles, and locks.
- The authoritative timer for session leases runs at the master, until a new master is elected the session lease timer is stopped; this is legal because it is equivalent to extending the client's lease.
- If a master election occurs quickly, clients can contact the new master before their local lease timers expire. If the election takes a long time, clients flush their caches and wait for the grace period while trying to find the new master. Thus, the grace period allows sessions to be maintained across fail-overs that exceed the normal lease timeout.

**Backup:**

- Every few hours, the master of each Chubby cell writes a snapshot of its database to a GFS file server in a different building.
- The use of a separate building ensures both that the backup will survive building damage and that the backups introduce no cyclic dependencies in the system.
- GFS cell in the same building potentially might rely on the Chubby cell for electing its master. Backups provide both disaster recovery and a means for initializing the database of a newly replaced replica without placing load on replicas that are in service.

**Mirroring:**

- Chubby allows a collection of files to be mirrored from one cell to another. Mirroring is fast because the files are small and the event mechanism informs the mirroring code immediately if a file is added, deleted, or modified.
- Provided there are no network problems, changes are reflected in dozens of mirrors world-wide in well under a second. If a mirror is unreachable, it remains unchanged until connectivity is restored. Updated files are then identified by comparing their checksums.

**Mechanisms for scaling:**

- There is just one master per cell and its machine is identical to those of the clients, the clients can overwhelm the master by a huge margin. Thus, the most effective scaling techniques reduce communication with the master by a significant factor.
- Approaches:
    - Create an arbitrary number of Chubby cells; clients almost always use a nearby cell to avoid reliance on remote machines. Our typical deployment uses one Chubby cell for a data centre of several thousand machines.
    - The master may increase lease times from the default 12s up to around 60s when it is under heavy load.
    - Chubby clients cache file data, meta-data, the absence of files, and open handles to reduce the number of calls they make on the server.
    - We use protocol-conversion servers that translate the Chubby protocol into less-complex protocols such as DNS and others.

**Partitioning:**

- Chubby's interface was chosen so that the name space of a cell could be partitioned between servers.
- If enabled, a Chubby cell would be composed of N partitions, each of which has a set of replicas and a master.
- Every node D/C in directory D would be stored on the partition P(D/C) = hash(D) mod N. Note that the meta-data for D may be stored on a different partition P(D) = hash($D_0$) mod N, where $D_0$ is the parent of D.

Goel, Ajay: Nu Id (001897443)

**Use, surprises and design errors:**

- **Use and behavior**
- **Java clients**
- **Use as a name service**
- **Problems with fail over**
- **Abusive clients**

- Chubby is a distributed lock service intended for coarse grained synchronization of activities within Google's distributed systems.

- It has found wider use as a name service and repository for configuration information.
- Its design is based on well-known ideas that have meshed well: distributed consensus among a few replicas for fault tolerance, consistent client-side caching to reduce server load while retaining simple semantics, timely notification of updates, and a familiar file system interface.
- We use caching, protocol-conversion servers and simple load adaptation to allow it scale to tens of thousands of client processes per Chubby instance.
- We expect to scale it further via proxies and partitioning. Chubby has become Google's primary internal name service. It is a common rendezvous mechanism for systems such as MapReduce. The storage systems GFS and Bigtable use Chubby to elect a primary from redundant replicas and it is a standard repository for files that require high availability, such as access control lists.