High-level Architecture

This lesson gives a brief overview of Chubby's architecture.

We'll cover the following

Chubby common terms#

Before digging deep into Chubby's architecture, let's first go through some of its common terms:

Chubby cell#

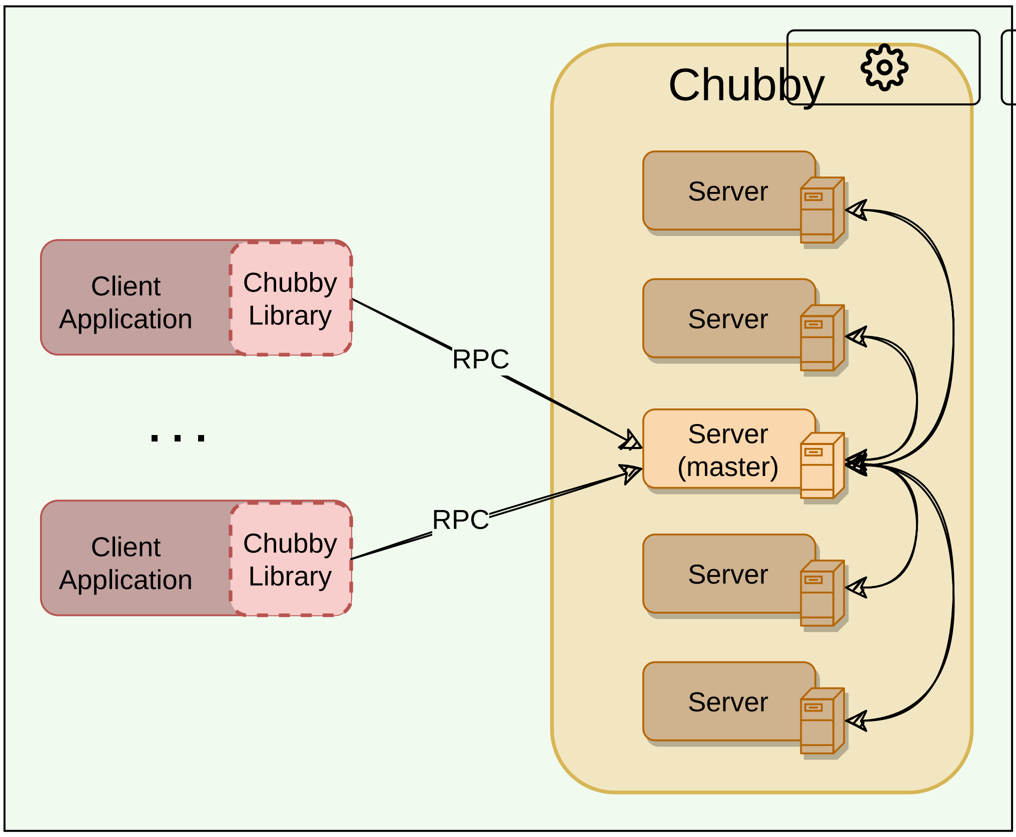
A Chubby Cell basically refers to a Chubby cluster. Most Chubby cells are confined to a single data center or machine room, though there can be a Chubby cell whose replicas are separated by thousands of kilometers. A single Chubby cell has two main components, server and client, that communicate via remote procedure call (RPC).

Chubby servers#

- A chubby cell consists of a small set of servers (typically 5) known as replicas.
- Using Paxos, one of the servers is chosen as the master who handles all client requests. If the master fails, another server from replicas becomes the master.
- Each replica maintains a small database to store files/directories/locks. The master writes directly to its own local database, which gets synced asynchronously to all the replicas. That's how Chubby ensures data reliability and a smooth experience for clients even if the master fails.
- For fault tolerance, Chubby replicas are placed on different racks.

Chubby client library#

Client applications use a Chubby library to communicate with the replicas in the chubby cell using RPC.



Chubby client library connects to Chubby master using RPC

Chubby APIs#

Chubby exports a file system interface similar to POSIX but simpler. It consists of a strict tree of files and directories in the usual way, with name components separated by slashes.

File format: `/ls/chubby_cell/directory_name/.../file_name`

Where `/ls` refers to the lock service, designating that this is part of the Chubby system, and `chubby_cell` is the name of a particular instance of a Chubby system (the term cell is used in Chubby to denote an instance of the system). This is followed by a series of directory names culminating in a `file_name`.

A special name, `/ls/local`, will be resolved to the most local cell relative to the calling application or service.

Chubby was originally designed as a lock service, such that every entity in it will be a lock. But later, its creators realized that it is useful to associate a small amount of data with each entity. Hence, each entity in Chubby can be used for locking or storing a small amount of data or both, i.e., storing small files with locks.

We can divide Chubby APIs into following groups:

- General
- File
- Locking
- Sequencer

General#

1. `Open()` : Opens a given named file or directory and returns a handle.
2. `Close()` : Closes an open handle.
3. `Poison()` : Allows a client to cancel all Chubby calls made by other threads without fear of deallocating the memory being accessed by them.
4. `Delete()` : Deletes the file or directory.

File#

1. `GetContentsAndStat()` : Returns (atomically) the whole file contents and metadata associated with the file. This approach of reading the whole file is designed to discourage the creation of large files, as it is not the intended use of Chubby.
2. `GetStat()` : Returns just the metadata.
3. `ReadDir()` : Returns the contents of a directory – that is, names and metadata of all children.
4. `SetContents()` : Writes the whole contents of a file (atomically).
5. `SetACL()` : Writes new access control list information

Locking#

1. `Acquire()` : Acquires a lock on a file.
2. `TryAquire()` : Tries to acquire a lock on a file; it is a non-blocking variant of Acquire.
3. `Release()` : Releases a lock.

Sequencer#

1. `GetSequencer()` : Get the sequencer of a lock. A sequencer is a string representation of a lock.
2. `SetSequencer()` : Associate a sequencer with a handle.
3. `CheckSequencer()` : Check whether a sequencer is valid.

Chubby does not support operations like append, seek, move files between directories, or making symbolic or hard links. Files can only be completely read or completely written/overwritten. This makes it practical only for storing very small files.

← Back
Chubby: Introduction

Next →
Design Rationale

✓ Mark as Completed

Report an Issue