Scaling Chubby

This lesson will explain different techniques that Chubby uses for scaling.

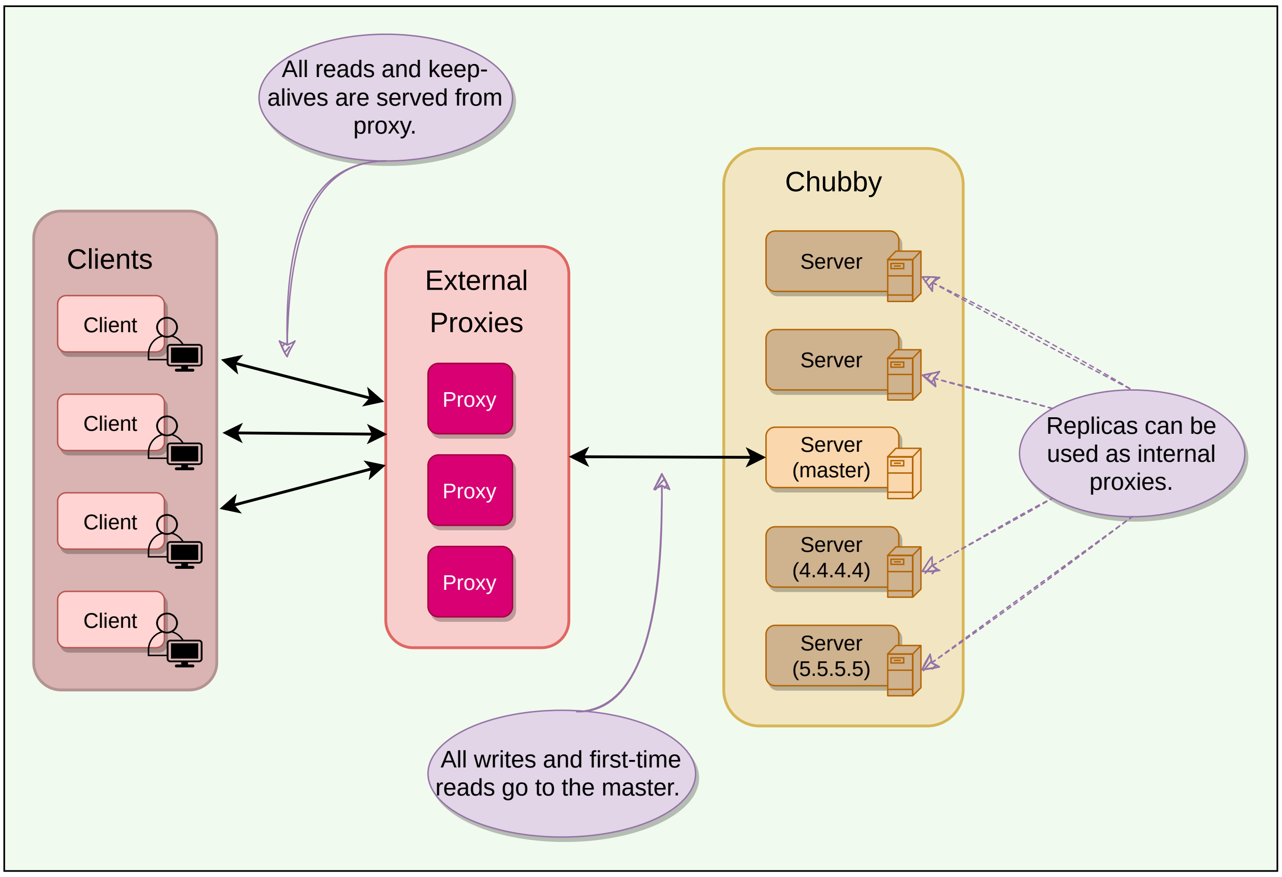
We'll cover the following

Chubby's clients are individual processes, so Chubby handles more clients than expected. At Google, 90,000+ clients communicating with a single Chubby server is one such example. The following techniques have been used to reduce the communication with the master:

- **Minimize Request Rate**: Create more chubby cells so that clients almost always use a nearby cell (found with DNS) to avoid reliance on remote machines.
- **Minimize KeepAlives load**: KeepAlives are by far the dominant type of request; increasing the client lease time (from 12s to 60s) results in less load on KeepAlive.
- **Caching**: Chubby clients cache file data, metadata, handles, and any absences of files.
- **Simplified protocol-conversions**: Add servers that translate the Chubby protocol into a less complicated protocol. Proxies and partitioning are two such examples that help Chubby scale further and are discussed below.

Proxies#

A proxy is an additional server that can act on behalf of the actual server.


All reads and keep-alives are served from proxy.
Clients
Client
Client
Client
Client
External Proxies
Proxy
Proxy
Proxy
Chubby
Server
Server
Server (master)
Server (4.4.4.4)
Server (5.5.5.5)
Replicas can be used as internal proxies.
All writes and first-time reads go to the master.

Chubby proxies

A Chubby proxy can handle KeepAlives and read requests. If a proxy handles 'N' clients, KeepAlive traffic is reduced by a factor of 'N.' All writes and first-time reads pass through the cache to reach the master. This means that proxy adds an additional RPC for writes and first-time reads. This is acceptable as Chubby is a read-heavy service.

Partitioning#

Chubby's interface (files & directories) was designed such that namespaces can easily be partitioned between multiple Chubby cells if needed. This would result in reduced read/write traffic for any partition, for example:

- `ls/cell/foo` and everything in it, can be served by one Chubby cell, and
- `ls/cell/bar` and everything in it, can be served by another Chubby cell

There are some scenarios in which partitioning does not improve:

- When a directory is deleted, a cross partition call might be required.
- Partition does not necessarily reduce the KeepAlive traffic.
- Since ACLs can be stored in one partition only, so a cross partition call might be required to check for ACLs.

Learnings#

Lack of aggressive caching: Initially, clients were not caching the absence of files or open file handles. An abusive client could write loops that retry indefinitely when a file is not present or poll a file by opening it and closing it repeatedly when one might expect they would open the file just once. Chubby educated its users to make use of aggressive caching for such scenarios.

Lack of quotas: Chubby was never intended to be used as a storage system for large amounts of data, so it has no storage quotas. In hindsight, this was naive. To handle this, Chubby later introduced a limit on file size (256kBytes).

Publish/subscribe: There have been several attempts to use Chubby's event mechanism as a publish/subscribe system. Chubby is a strongly consistent system, and the way it maintains a consistent cache makes it a slow and inefficient choice for publish/subscribe. Chubby developers caught and stopped such uses early on.

Developers rarely consider availability: Developers generally fail to think about failure probabilities and wrongly assume that Chubby will always be available. Chubby educated its clients to plan for short Chubby outages so that it has little or no effect on their applications.

← Back
Database

Next →
Summary: Chubby

✓ Mark as Completed

Report an Issue