Chubby Distributed Lock System

Evan Wireman, wireman@wisc.edu
Aanandita Dhawan, dhawan6@wisc.edu
Safi Nassar, nassar2@wisc.edu
CS, UW–Madison, WI

ABSTRACT

In this project, we implemented Chubby, a distributed, strongly consistent lock system [1]. We were able to implement most of the functionality, foregoing the client notification system due to the complexity of that feature. We provide a ‘app::client‘ name-space, which provides clients with a means to communicate with the lock system. For leader election and write/delete consensus, we used NuRaft, an open-source project from eBay. Our design provides recovery for both client and server fail-over, concurrent write requests, and strongly consistent master election. In this paper, we will discuss our design choices and highlight some of the capabilities of our implementation.

1 DESIGN

Our implementation provides a client library, which one could use to communicate with the lock system through a series of methods. In order to establish a session, the client must run client::create_session(), which will loop through the nodes in the Chubby cell, stopping once the master is found. Once a session is established, a session maintenance thread is detached from both the client and the server. This thread is responsible for sending keep_alive rpcs back and forth, ensuring the continued vitality of the session. The server will receive a keep_alive, wait until it is within one second of the lease expiration, then reply to the client. Upon receiving a reply, the client will immediately send another rpc to the server.

If a client does not hear back from the server by its lease expiration, it enters a state of jeopardy. Once every second (chosen arbitrarily), the client will send a keep_alive rpc to every server, except this time it packs its list of locks it was holding during the session. Once a new master is found, the master will receive this keep_alive and attempt to continue the client's session. If every lock the client held in the previous session is available, the session is restored, otherwise the server lets the client know the session has expired. We used values directly from the Chubby paper for lease extension, the initial lease length, and jeopardy duration.

For tracking lock states and active sessions, we use in-memory data structures. The content of each lock is stored in files on disk. If a client creates a lock at path /foo, the server would create an empty file on disk at path ./tmp/SERVER_RAFT_ID/foo. The server would also create in memory data structures to track the existence, state, and owner/s. Interactions with locks generally require scanning ./tmp/SERVER_RAFT_ID to ensure the existence (in the case of read, write, acquire, release, and delete) or non-existence (in the case of create). Additionally, "privileged" operations (read, write, release, delete) require scanning the in memory data structures to ensure the client owns the lock at the proper exclusivity. All operations with the lock service require a check that the client has an active session with the server.

For propagating changes and ensuring consensus, we use NuRaft, an open source implementation of Raft [2] designed and maintained by ebay. After doing some research, we found that this was the best library for our purposes. It was simple to use, easy to integrate, and handled leader election behind the scenes. All we had to change was adding the ability to log file creates, writes, and deletes, and ensure that, upon committing, a file is created/deleted/written to disk.

2 RESULTS

We have evaluated the implementation on clusters of different sizes, ranging from 1 to 10 nodes.

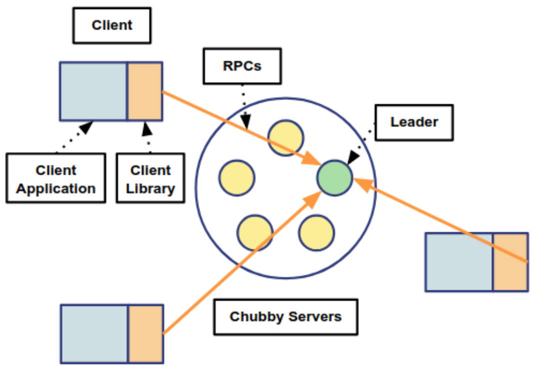


Figure 1: Chubby Architecture

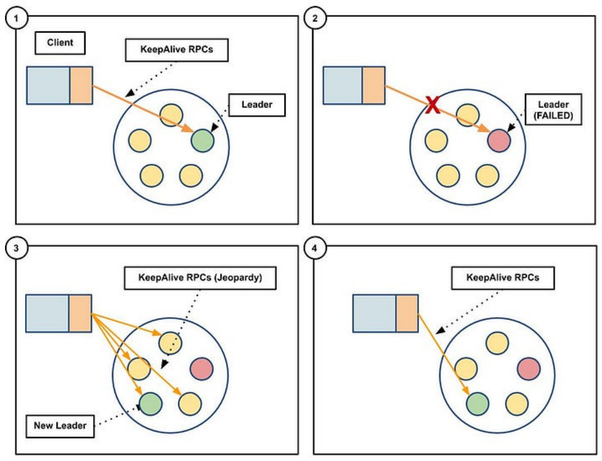
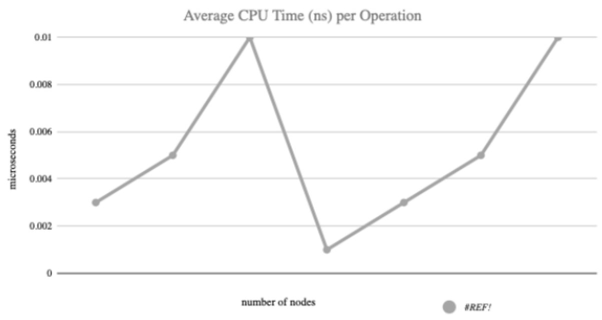


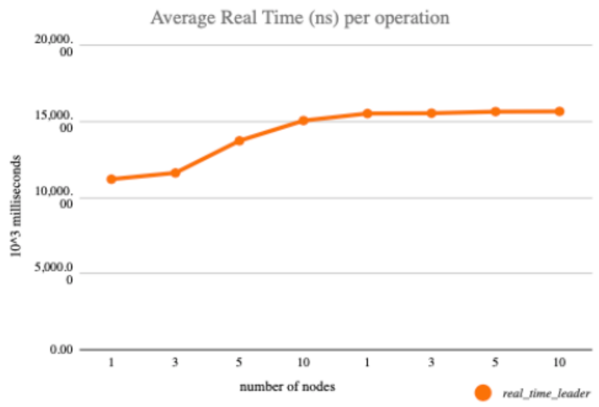
Figure 2: Jeopardy Process

2.1 Time Metrics

(1) Average CPU time (ns) per operation

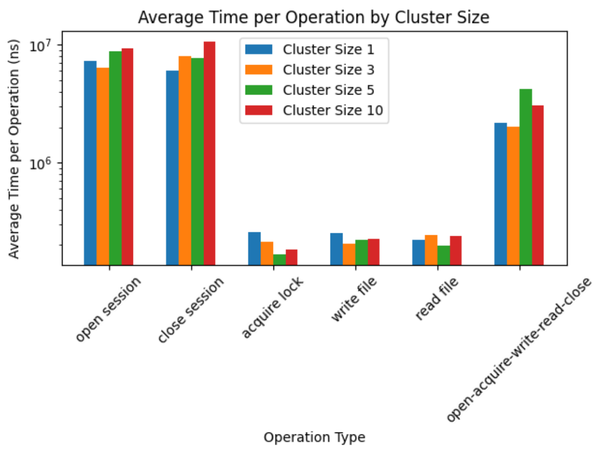


(2) Average Real time (ns) per operation



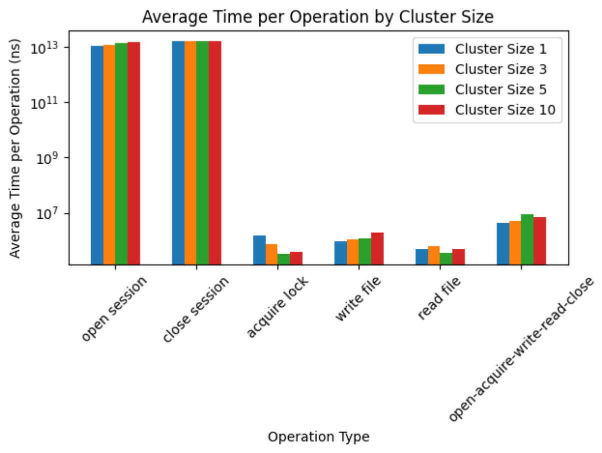
(3) Average Time per Operation by Cluster Size (CPU Time)

List of Operations : Open session, Close session, Acquire lock, Write file, Read file, Open-acquire-write-read-close
Please note that the graph values are plotted in the logarithmic scale for this average time plot!



(4) Average Time per Operation by Cluster Size (Real-time)

List of Operations : Open session, Close session, Acquire lock, Write file, Read file, Open-acquire-write-read-close
Please note that the graph values are plotted in the logarithmic scale for this average time plot!



3 TAKEAWAYS

Reflecting on the lessons learned, one key aspect is the in-depth understanding of Chubby's architecture. By studying Chubby's design principles and implementation details, valuable insights were gained that could be applied to future development efforts. Another important lesson was the ability to tailor open-source projects such as NuRaft and msd channels to align with the specific requirements of the project. This customization ensured that the selected tools and libraries effectively met the project's needs and integrated seamlessly into the overall system. Lastly, adopting a test-first development approach proved to be beneficial, particularly for projects with ample potential for bugs and errors. By prioritizing thorough testing and creating test cases before implementing new features or making modifications, the project was able to identify and address issues early on, resulting in a more robust and reliable final product.

REFERENCES

[1] Mike Burrows. 2006. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*. 335–350.

[2] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 305–319.