Lock Service *(simple Chubby)*

Distributed Lock Service ; NuRaft ; In-memory log;
Persistent State Machine;

Aanandita Dhawan
dhawan6@wisc.edu

Evan Wireman
wireman@wisc.edu

Safi Nassar
nassar2@wisc.edu

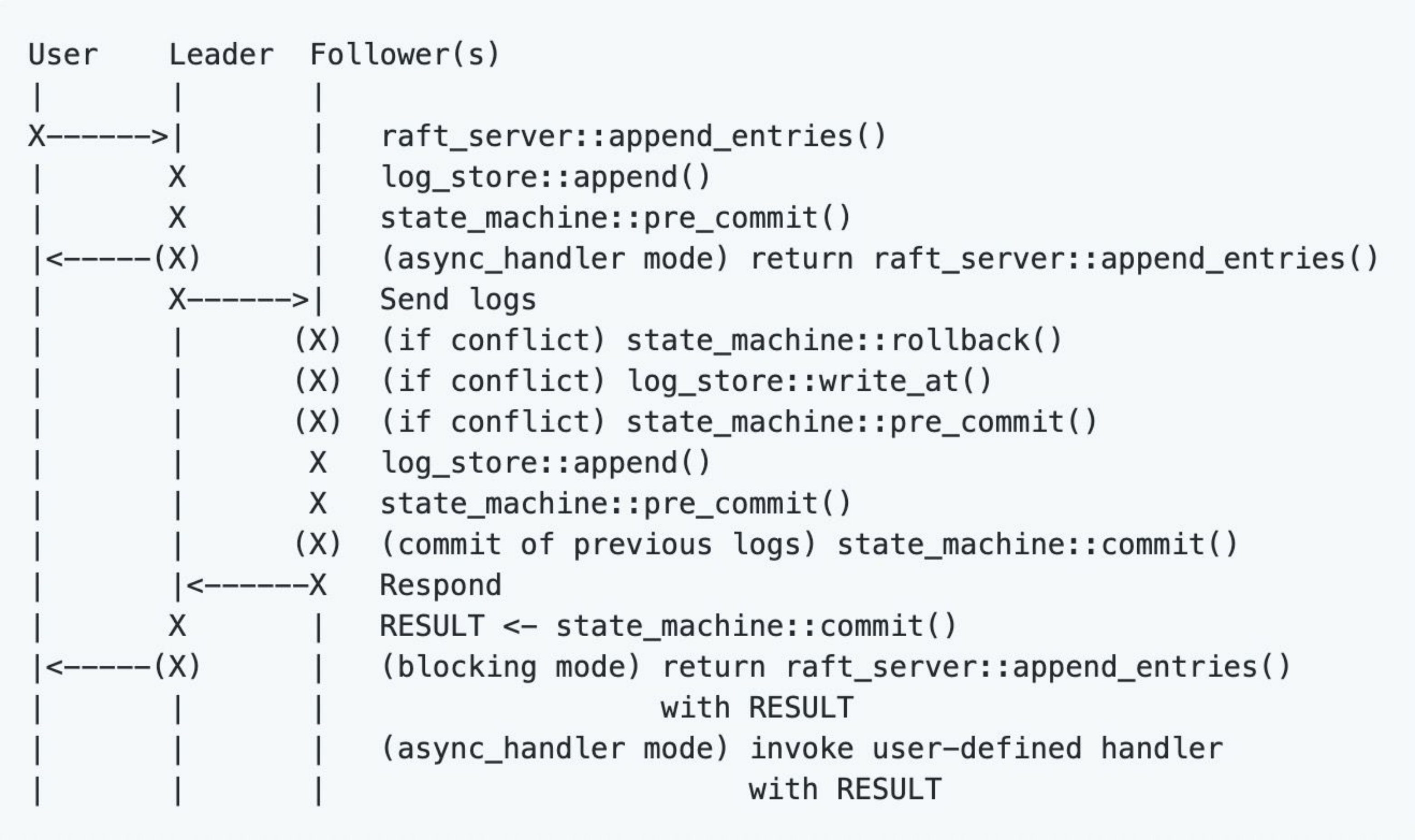
Design Decisions

- NuRaft for consistency
  - Open source, simple to implement, easy to import, leader consensus done behind the scenes
- No subscription to events
  - Would have required complicated heartbeat RPC responses
- msd.channels.h was used for inter-thread communication
  - Blocking channel implementation, open source
- Single binary, separate namespaces for server and client
  - A Client can include app.h and have access to client methods

NuRaft

- NuRaft by ebay (open source) for consensus
- Easy to configure and install
- Simple API (get, set)
- Shrouds leader election
- Did not require much code alteration to be able to log [file_name, contents] pairs.
- Asynchronous mode (we used synchronous for simplicity)

NuRaft Flow



```
User    Leader  Follower(s)
|       |       |
X------>|       |   raft_server::append_entries()
|       X       |   log_store::append()
|       X       |   state_machine::pre_commit()
|<-----(X)      |   (async_handler mode) return raft_server::append_entries()
|       X------>|   Send logs
|       |      (X)  (if conflict) state_machine::rollback()
|       |      (X)  (if conflict) log_store::write_at()
|       |      (X)  (if conflict) state_machine::pre_commit()
|       |       X   log_store::append()
|       |       X   state_machine::pre_commit()
|       |      (X)  (commit of previous logs) state_machine::commit()
|       |<------X   Respond
|       X       |   RESULT <- state_machine::commit()
|<-----(X)      |   (blocking mode) return raft_server::append_entries()
|       |       |                   with RESULT
|       |       |   (async_handler mode) invoke user-defined handler
|       |       |                        with RESULT
```

Structs

```
struct Lock {
   std::string path;                                     // Path to the file
   LockStatus status;                                    // Is it shared or exclusive?
   std::shared_ptr<std::map<std::string, bool>> owners;  // Who owns the lock
   std::string content;                                  // File content
 };

struct Session {
   string client_id;                                // Id of the client with whom this session exists
   chrono::system_clock::time_point start_time;     // Start time of the local session
   chrono::milliseconds lease_length;               // Length of the session lease
   shared_ptr<msd::channel<int>> block_reply;       // Channel used for blocking the reply to keep_alive rpcs
   shared_ptr<map<string, shared_ptr<Lock>>> locks; // Locks acquired with the session
   bool terminated;                                 // Indicator of if the session has been terminated manually
}

enum LockStatus {
   EXCLUSIVE,
   SHARED,
   FREE
}
```

Data structures *(singleton pattern)*

Server:

```
map<file_path, struct Lock> locks;
map<client_id, struct Session> sessions;
```

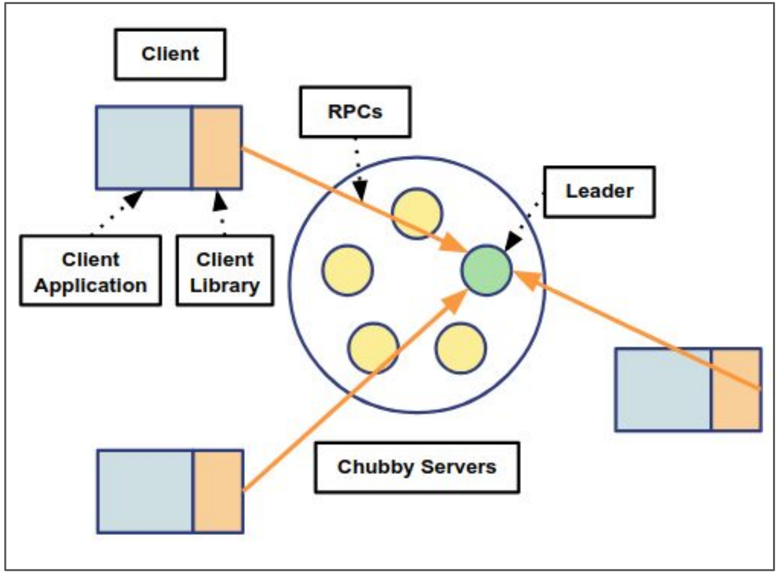
Client

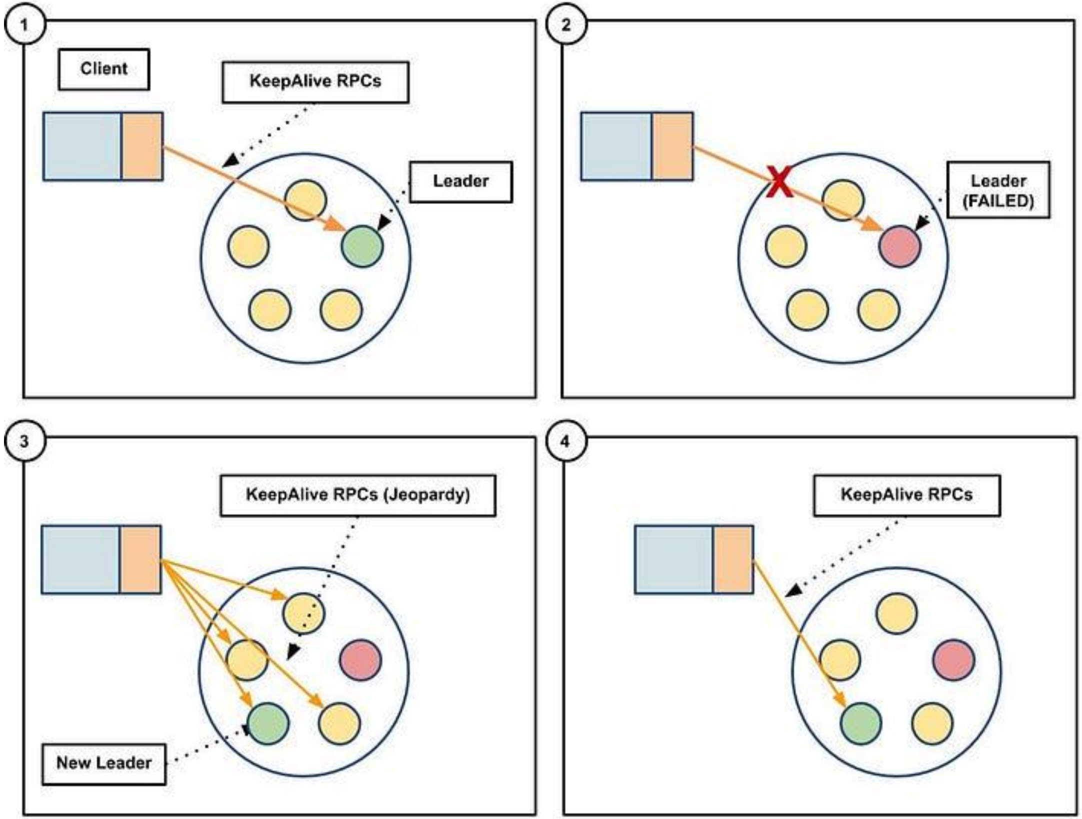
```
string session_id;
chrono::system_clock::time_point lease_start;
chrono::milliseconds lease_length;
shared_ptr<map<string, LockStatus>> locks;
bool jeopardy;
bool expired;
shared_ptr<Node> master; // A Node is just an abstraction for rpc endpoint
```

RPCs

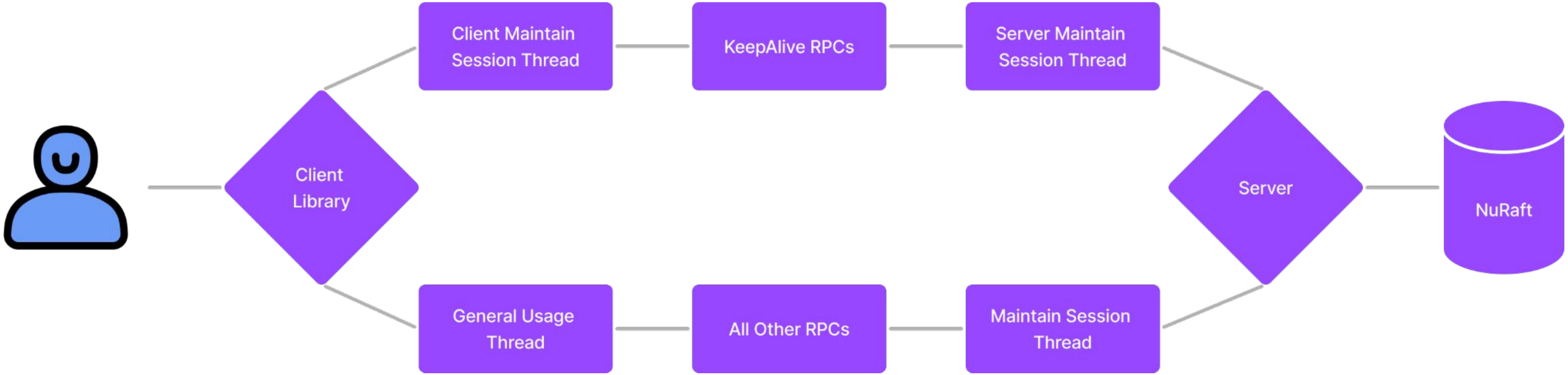
- Init_session(client_id)
- Close_session(client_id)
- Keep_alive(client_id, time_point)
- Keep_alive(client_id, time_point, map<file_path, LockStatus>)
- Open_lock(client_id, file_path)
- Delete_lock(client_id, file_path)
- Acquire_lock(client_id, file_path, LockMode)
- Release_lock(client_id, file_path)
- Read(client_id, file_path)
- Write(client_id, file_path, content)

Chubby Architecture





Our Project’s Architecture



Live Demo

We will show our implementation, in a 5 node cluster:

- Create and maintain sessions (heartbeats)
- Handle server failures (client in jeopardy)
- Handle client failures (session timeout)
- Open/close locks
- Acquire/release locks
- Read/write to locks

Takeaways

Future work:

- Event subscription
- Evaluation using raft vs. paxos for consensus
- Comparison with ZooKeeper

Lessons Learned:

- The architecture of Chubby in great detail
- Tailoring open source projects (NuRaft, msd channels) to match our needs
- Test-first development can be useful for projects that has a lot of room for bugs