# Powering ChiselStore with Omni-Paxos

VASIGARAN SENTHILKUMAR

26.03.2022

## 1 Introduction

Omni-Paxos is a replicated log library of sequence consensus similar to Raft. It is used to build consistent services such as replicated state machines. It provides increased resilience and performance compared to raft.Omni-Paxos differs from Raft in the ballot leader election. In Raft ,a server must have an up to date updated log to become a leader. This condition is relaxed in Omni-Paxos , any server can become the leader. The log gets synchronised during the prepare phase.

The project aims at replacing Raft with Omnipaxos in an Open source system called Chiselstore. Chiselstore is a distributed SQLite which is powered by Raft consensus algorithm implemented in Rust language. Replacing Raft with 0mnipaxos changes the sequence consensus algorithm and involves removing the raft dependency and replacing it with Omnipaxos core.

## 2 Design

The design structure was first aimed at understanding how the raft implementation of Chisel store works. So the execution was traced starting from the gouge.rs. The gouge is the distributed SQL server that was built on ChiselStore and RPC. Then after identifying the design , the equivalent implementation design in terms of Omnipaxos was formed. We also use the Omnipaxos core library implemented in Rust. It implements the algorithms of Omnipaxos as plain Rust structs and is suitable for integration with systems that already have an async runtime or are implemented in an actor framework.

Remote Procedure Call (RPC) is our network transport layer. The client inputs the command to execute through the RPC from the gouge. The communication between the peers of each StoreServer happens through RPC. The RPC forwards the the messages to the server. The proto RPC is used here to define the interface for the RPC protocol.

The StoreServer is main server as part of Chiselstore. Storeserver contains instance of Sequence Paxos , Ballot leader election and store. The SQL statement that the client provides is replicated in the store as Store Command and the leader will propose the entry to the peers.

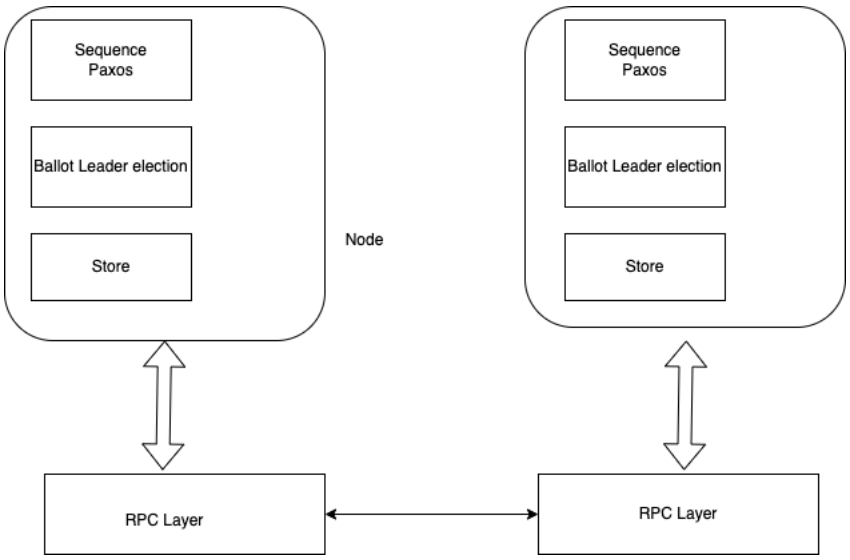


Figure 1: Design Diagram .

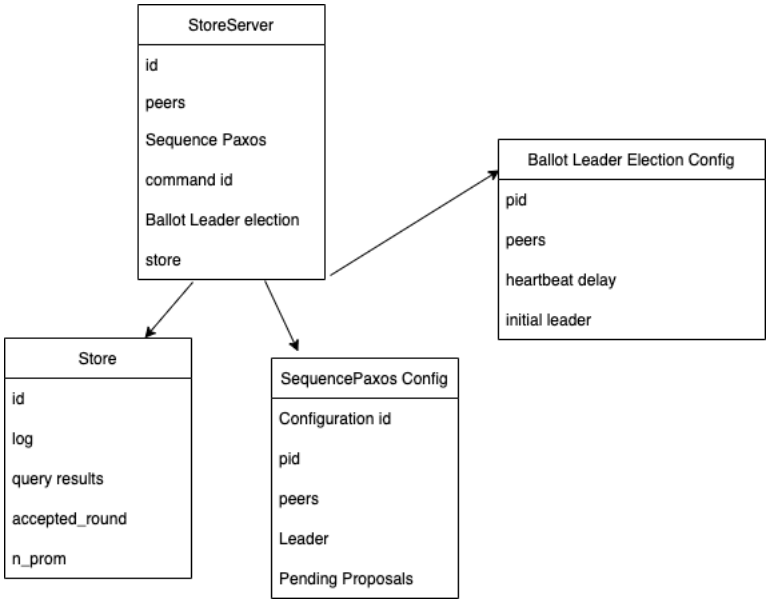


Figure 2: Class Diagram .

# 3 Implementation

The solution was implemented starting from the RPC interface, modifying the proto.proto in order to match the message structure of Omnipaxos. Each type of Paxos Message was to given a structure. The different paxos messages were Prepare, Promise, AcceptSync, FirstAccept, Accept decide, Accepted, Decide, Proposal Forward,Compaction, Forward Compaction, Accept Stop Sign, Accepted Stop Sign,Decide Stop Sign, BLE Message, Heart beat Request , Heart beat reply. The other components of the message struct like the ballot, stop sign has to be defined. The service RPC is provided to generate a RPC client service. The proto file compiles into proto.rs.

The next part was handling the RPC messages. The outcoming messages from the server needs to be converted into the RPC interface format and when it is received the RPC message has to be converted back to sequence Paxos. I defined the function RPCtoSeqPaxos which inputs the RPC message and converts into equivalent Sequence Paxos Message form declared in messsages.rs of Omnipaxos core. Since Each Paxos Message can be of various type, declared various utility functions to convert into equivalent Sequence paxos.

Similarly the function SeqpaxosToRpc converts the Sequence Paxos Message to the RPC message form mentioned in the RPC proto for network Transport. The same conversion is being done for Ballot Leader election Messages which consists of heart beat requests and heart beat replies.

The trait Store Transport is implemented to Provide networks access to the chisel store server. We define the functions of store Transport in RPC file. Then a connection is made to the RPC which can provide query results to the client or handle the sequence paxos message and the ballot leader election messages and provide to the server.

The next was to implement the StoreServer which is the server of the chisel store. Had to replace all the raft configurations with that of sequence paxos from the Omnipaxos core. The Storeserver will now consists of instance of sequence paxos, ballot leader election and a store. Also there is are notifiers for both sequence paxos message and ballot leader election message using a crossbeam channel which is the multi producer and multi consumer channel for message passing. There is a an instance of QueryResultsHandler which is a struct that consists of Notifiers and vector of query rows in the form of a hash map. The query results store the results of queries till the decided index.

The function start implements a store server. In the function Run we run the blocking event loop with a delay of 1 ms. In the run function we handle any incoming sequence paxos message or the ballot leader election message and provide it to the instance of sequence paxos. For this we use the handle function provided in sequence paxos.rs of the Omnipaxos core. When a leader is elected in the ballot , we handle leader in sequence paxos. The the outgoing messages are provided to Store Transport functions and then provided to RPC layer via the SendSeqPaxosMsg ,SendBleMessage functions in RPC.rs.

Then the aynchronous query function was modified to execute any incoming query through the RPC. So the incoming sql statement is changed to a Store

Command. We append the store command to the sequence Paxos. There is a notifier added here to wait for the sequence paxos to return the query results. When the query results are return, it is provided to the RPC layer for delivery to the client.

Each Store server maintains a store . The store contains the log of sequence paxos which includes Last promised round,Last accepted round,Length of the decided log, query results.

Then there is the storage for the Omnipaxos. This storage uses the function from Omnipaxos core to append to log, retrieve from log , setting the promise, setting the decided index in the log and executing the queries, setting the accepting round, getting accepted round etc.. While setting the decide index we run the sql queries to the decided index and add the results to the query results of the Store.

There were quite a few challenges during implementation. Adapting to the Rust programming language was challenging. So the conversion of sequence paxos message to RPC message and vice versa was not easy. But the structure of how Raft was handling it was similar. In the server file, the implementation of the store server the little raft functions had to be changed with the Omnipaxos core functions.

# 4 Testing

The first test involves starting 3 nodes with each node having the other 2 as peers and checking if the distributed sql server functionality works. This to validate the Omnipaxos configuration was successful. And guarantees the properties of Sequence consensus of Validity , uniform agreement, integrity and termination. The Quorum based leader election correctness property can also be verified. Since the quorum connected server is only elected and a monotonically increasing unique ballot can be observed from the logs. Also the important property of quorum connected eventual accuracy is verified.This time since all quorum connected servers are connected to each other, the leader with the highest ballot is elected.

```
cargo run --example gouged -- --id 1 --peers 2 3
cargo run --example gouged -- --id 2 --peers 1 3
cargo run --example gouged -- --id 3 --peers 1 2
```

The next test was testing to see the most important improvement of Omnipaxos over Raft. Started 3 nodes but only one quorum connected server. The quorum connected server was elected the leader. Because that quorum connected server with the id 1, received the majority of heart beat replies and elect it is as only leader.

```
cargo run --example gouged -- --id 1 --peers 2 3
cargo run --example gouged -- --id 2 --peers 1
cargo run --example gouged -- --id 3 --peers 1
```

The next test was having multiple quorum connected servers but the quorum connected servers are not connected to each other. But they are connected to a majority. Any majority overlaps in at least 1 server because of the quorum property.

```
cargo run --example gouged -- --id 1 --peers 2 3
cargo run --example gouged -- --id 2 --peers 1
cargo run --example gouged -- --id 3 --peers 1
cargo run --example gouged -- --id 4 --peers 3 5
cargo run --example gouged -- --id 5 --peers 4
```

In this case node 4 was elected as the leader. both nodes 1 and 4 had a majority but were not connected to each other. But 4 had the highest ballot and hence was chosen as the leader. This also verifies quorum connected eventual accuracy.

# 5 Future Work

There was no time to implement re configurations. It would have been nicer if there was a little bit more time for implementing it. Also in the testing, the partial connectivity property of Omnipaxos could have been done.

# 6 Summary

The most important problem was initial understanding how Raft was in place in the chisel store. I had some discussions with fellow students in understanding the code and how the implementation was done and what changes would be needed to replace with Omnipaxos. The syntax of Raft took some time to get used to. A lot of stack overflow also helped.