ChiselStore: replacing Little Raft with OmniPaxos

Chrysoula Dikonimaki

27/3/2022

1 Introduction

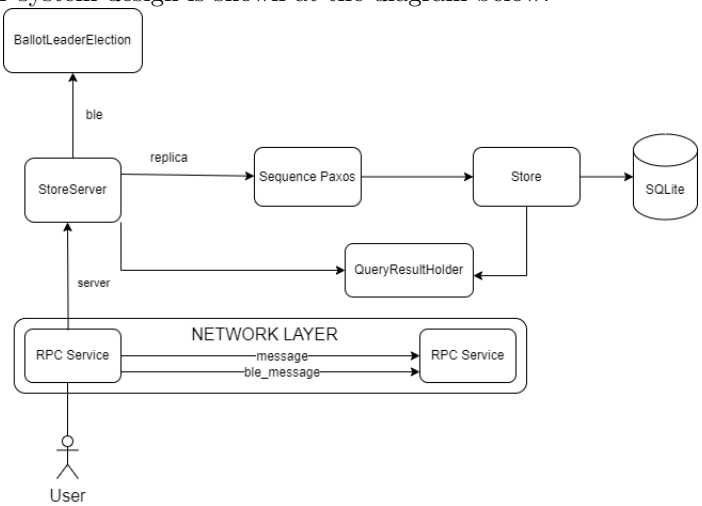
ChiselStore is an an embeddable, distributed SQLite for Rust, powered by Little Raft. The aim of this project is to replace OmniPaxos with Little Raft for an open-source project, called ChiselStore. OmniPaxos is a replicated log library which aims to hide the complexities of consensus to provide users a replicated log that is as simple to use as a local log. Thus, our goal is to make ChiselStore simpler and more efficient by implementing it with a new and promising library.

For this assignment, I collaborated with Olivia Höft.

The implementation can be found here.

2 Design

Our system design is shown at the diagram below:



I tried to keep the design similar to the original ChiselStore design. I added the QueryResultHolder so I can hold the query's results and the notifiers (notify

when the result is saved). Moreover, the Store is now moved inside the SequencePaxos replica because it has the Storage. Also, we don't have a StateMachine and a Cluster.

3 Implementation

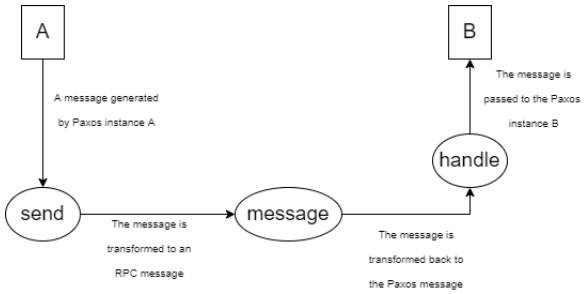
3.1 Implementation Details

According to the OmniPaxos protocol all you have to implement is two things: 1) the network 2) the storage, so the first change we did was to change the /proto/proto.proto file. We used the proto3 version do the protocol bugger language to define our network protocol. In order to do it, we used Prost, a Protocol Buffer implementation for Rust, as the original ChiselStore does. We defined all OmniPaxos messages in the proto file and in order to use them for RPC we defined our service. Our service has now only 3 methods: one for the BLE message, one for Paxos messages and the execute method, which remains the same.

It is important to mention here that when the send method is being called because it is an important part of our implementation. When somebody starts the server then they have to start 2 threads, one for the run method and one for the run_leader method. These methods are very important. The first one is the one which called periodically the get_outgoing_msgs for both paxos and ble and call the send and send_ble periodically. The second one calls the handle_leader for the paxos.

The next step was to change the rpc.rs file to use the new RPC service. The main usage of this file is to transform Paxos messages to Rpc messages and back. This way we can send the messages that a Paxos instance creates over the network to the other Paxos instances which are located to the other servers and when the others get them they will pass them back to their own Paxos instance. So the method which takes the Paxos message and transoforms it to the Rpc message is the send. When the Paxos instance generates an outgoing message we call the send method which actually transforms the message to an RPC message and makes an RPC call to the server that the message should be send to. At this RPC call we actually call the message method, which is responsible to transform the message back to a Paxos message and pass it to the sequence Paxos instance. This is done by calling handle in the server. The We follow the similar logic for Ble messages. The corresponding methods for Ble are the send_ble, ble_message and handle_ble.

The flow of sending a message from a replica instance A to B is described in the following image. The arrows are the message types and the circles describe the methods that are being called. It is a simplified diagram aims to show us the flow of the message sending.



The next step was to implement the storage, as the documentation says. We implement the storage as a part of the Store.

3.2 Query execution

When the user wants to execute a query they make an RPC call by calling the execute method. This method calls the server's query method to pass the query to the server. At the server's query method we call the Paxos' append method in order to append the command to the log. The purpose of doing this is because we want the command to be decided from the Paxos protocol so we can run it and return the result to the user. So after appending it we wait for the command to be decided. In order to do this, we use a Notify and we wait to be notified that the query ran and the result of it is saved. OmniPaxos will call the set_decided_idx at the storage, when a new index is decided which means that some extra command has been decided. At this method, we get the new decided commands, run the queries that these commands hold, save the result to QueryResultHolder and notify that the command has been decided. After notifying that the result is saved the method that was waiting (the query method) gets the result from the QueryResultHolder and returns it back to the user.

3.3 Extra Features

We added Reconfiguration and Snapshotting. However, none of them was fully implemented in OmniPaxos so more details about can be found at Furute Work

3.4 Challenges

The most challenging part of the assignment was to understand the structure of the original code. I haven't used Rust before so this made it more challenging. Also, writing the network layer needed a lot of time.

4 Testing

In order to test our system, we wrote some Integration tests. The main structure of them is the following: set up the system with a specific number of nodes,

execute some queries to them and finally shut them down.

The command to run a specific test is: cargo test testNo which No is 1,2.. etc.

The first test is just a simple test that we just run a simple query "SELECT 1+1;" just to see that we can actually decide on a command and run it.

```
test test1_simple_query ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 2 filtered out; finished in 16.32s
```

The second test is a complete test that contains the following sequence of commands: create a table, insert a value on it and then select and check that the value is correct. All these calls are made to the node with id 1.

```
test test2_create_insert_select ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 2 filtered out; finished in 16.29s
```

The third test is similar to the second one but we ran it with one more node in the system and we run the last command, the select command to second node. We select from the 2nd node even if we send the insert to the first one because we want to be sure that what is decided is decided to all nodes.

```
test test3_create_insert_select ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 2 filtered out; finished in 16.79s
```

The fourth test is commented because it's the reconfiguration. More details can be seen bellow.

The fifth test is a test which will help us ensure that the nodes decide the same sequence of command.

```
test test5_decide_same_sequence ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 4 filtered out; finished in 16.20s
```

The sixth test is a test that we want to see that if the majority of nodes is alive (for example, after killing some) it will continue working.

```
test test6_kill_a_node ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 4 filtered out; finished in 18.78s
```

The seventh test is a test that checks that the Snapshotting works. Extra assertions need to be added.

```
test test7_snapshotting ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 5 filtered out; finished in 16.34s
```

Moreover, a bat file can be found under the /examples directory which sets up 3 nodes and a command line to write queries. An execution example is shown below:

```
gouge=# CREATE TABLE TEST1 (T INTEGER);
gouge=# INSERT INTO TEST1 (T) VALUES (1);
gouge=# SELECT * FROM TEST1;
["1"]
gouge=#
```

5 Future Work

The code, especially, the rpc.rs file, needs refactoring because the same code is repeated again and again so this is something that can be done in the future.

Also, I implemented the code for the reconfiguration and one test to test it but then I got the following error:

```
---- test4_reconfiguration stdout ----
thread 'test4_reconfiguration' panicked at 'not yet implemented: forward stopsign', C:\Users\Chrysa\.cargo\git\checkouts\omnipaxos-
3d15a851294b8abd\36f6913\omnipaxos_core\src\sequence_paxos.rs:488:22
```

As far as I understand reconfiguration is not fully implemented at the OmniPaxos so I just commented the code. If we try to reconfigure from the leader

and not from a follower it probably works though because it is clear that the message is that is not implemented is the "Forward StopSign" which forwards it to the leader. Uncomment the code and run the test for it when the OmniPaxos reconfiguration is completed it is something that should be done in the future if we want ChiselStore to have this functionality as well. For testing purposes, I just added a reconfigure method in ServerStore which was supposed to call the reconfigure method at the Paxos but practically, if we want to use it in practise we should have an Rpc method so the user could call it. This is also something that could be done in the future.

6 Summary

To sum up, at this project we replaced Little Raft with OmniPaxos in order to implement a distributed database, called ChiselStore. The main functionality works as excepted. Also, extra features added (Reconfiguration, Snapshotting) which hopefully, will work as expected after they are implemented in Omni-Paxos. Sufficient integration tests have been written to automatically test the whole system and its properties.