SIJIA ZHU
Assignment2 overview

For this assignment, we implement the CURD operation for multi-thread client-server communication through RPC. Through this implementation, we can understand the idea how the RPC work and how the different between RPC and socket. We also learned how different between the RPC frameworks, there gRPC, Thrift, RMI, etc. I do a research between gRPC and thrift and make a decision to use gRPC as start point for my implementation. We learned how to handle multi-thread in Key-Value CURD operation.

Technical impression

This project is well structured and good practice for us understand gRPC framework and Multi-thread protocols.

First, I learned gRPC framework when I decide to choose between gRPC and Thrift. The different between gRPC and thrift is like different between Google and Facebook, why I choose gRPC is related to HTTP/2.0 message which is the future of HTTP standard. We can have much more standard framework to cooperation when in a group project. In gRPC we use protocol buffers to communicate between client and server and gRPC use channel to connect server on specified host and port and is used when creating a client stub("client"). Clients can specify channel arguments to modify gRPC's behavior.

Second, I learned how to use multi-thread in gRPC and handle CURD operation in multi-thread. There are several way to use mulit-thread in gRPC, we can use threadpool and semaphore. I use semaphore to simulate multi client to access multi server. As semaphore, we can precise control how to use thread, when we want to start use new thread, we need to acquire new thread from system, and we need to release thread after we don't need it. We can limit the number of thread to use to compare the speed different between single thread and multi-thread operation. We can see the result below

```
Client Thread Number   Server Thread Number  Test Running Time   Result
100                       1                    5                   18.0 RPCs/s
100                       5                    5                   108.4 RPCs/s
100                       10                   5                   217.6 RPCs/s
100                       100                  5                   1688.4 RPCs/s
```

When there is 100 client access one thread for running 5 second, it only can have 18 RPC channel communication for one second, but if we increase to 5 server thread, the result become 108.4 RPCs/s which is 5 time and it is linear increase when we increase the number of server thread which can make efficient when we use multi-thread to CURD operation. In gRPC client, there is method called ListenableFuture which is concurrent method that it allow multi channel to communicate the server and the listener in this method will give the result callback function that when the server response so it don't need to wait a queue when other chancel is running in the server. Which we can increase the speed for asynchronous without blocking

current thread. In the server, I also use asynchronized way to handle multi-thread which means for each CURD operation as PUT-GET-UPDATE-DELETE, I will block other thread to access CURD operation. I use ConcurrentHashMap to access storage without worry about save wrong result.

Finally, we design how to test in this multi-thread client and server operation. I use single thread in client side but I make random operation in while loop which it will continuously to do CURD operation randomly. You can see code below:

```
70    void doClientWork(AtomicBoolean done) throws InterruptedException {
71        Random random = new Random();
72        AtomicReference<Throwable> errors = new AtomicReference<>();
73
74        while (!done.get() && errors.get() == null) {
75            // Pick a random CRUD action to take.
76            int command = random.nextInt(4);
77            if (command == 0) {
78                doPut(channel, errors);
79            } else if (command == 1) {
80                doGet(channel, errors);
81            } else if (command == 2) {
82                doUpdate(channel, errors);
83            } else if (command == 3) {
84                doDelete(channel, errors);
85            } else {
86                throw new AssertionError();
87            }
88        }
89        if (errors.get() != null) {
90            throw new RuntimeException(errors.get());
91        }
92    }
93
```

For each operation, I use semaphore and ListenableFuture to no block other operation access the server. In another words, I use single thread to simulate multi client access server and then I set semaphore number in server to make sure server is multi thread, we just can see the result as mention before, multi thread server have faster communication way. And I record log for each operation in both client and server, if we find some FAIL operation, we can tracking client Id or and Server Id to know why it FAIL.For example, you can see log below

```
Client log: Feb 27,2020 20:21:38:184 CLIENT_ID: 738 SEND: "DELETE" KEY-> -98
Server log: Feb 27,2020 20:21:38:185 CLIENT_ID: 226 -> SERVER_ID: 24 RECEIVE: "DELETE" KEY-> -98
Server log: Feb 27,2020 20:21:38:210 SERVER_ID: 26 -> CLIENT_ID: 223 SEND: "UPDATE" KEY-> -98 VALUE-> 83
Client log: Feb 27,2020 20:21:38:212 SERVER_ID: 26 -> CLIENT_ID: 223 RESPONSE: "UPDATE" SUCESS KEY-> -98 VALUE-> 83
Server log: Feb 27,2020 20:21:38:236 SERVER_ID: 22 -> CLIENT_ID: 224 Throw Exception: "UPDATE" FAIL KEY-> -98 NOT FOUND
Server log: Feb 27,2020 20:21:38:236 SERVER_ID: 24 -> CLIENT_ID: 226 SEND: "DELETE" KEY-> -98
Client log: Feb 27,2020 20:21:38:237 Server Throw Exception ->  CLIENT_ID: 224 RESPONSE: "UPDATE" FAIL KEY-> -98 NOT FOUND
Client log: Feb 27,2020 20:21:38:237 SERVER_ID: 24 -> CLIENT_ID: 226 RESPONSE: "DELETE" SUCESS KEY-> -98
```

We can find DELETE first and UPDATE is after, then UPDATE throw exception send back to server because Key is not found.