SIJIA ZHU

Youtube Link: https://www.youtube.com/watch?v=S5wqJ9vk3ps

Assignment3 overview

For this assignment, we implement the CURD operation for multi-thread server for 5 different instances of server through RMI, through this implementation, we can understand the idea how RMI work and how the different server communication. We learned how to data consistency through Two-phase commit protocol.

Technical impression

This project is well structured and good practice for us understand RMI framework and Two-phase commit protocol.

First, there is some tricky story in RMI framework. RMI use stub represent remote object controlled in local. If you don't understand how the caller invoke a method is responsible for carrying out the method and call on the remote object, you might lose in path. For example, in my server class, I use myPort to represent the current port number, when in two phase commit protocol, I registry the master server and send it to another node server ack function through stub. If you don't know the myPort will switch based on who is the object of the function, you might feel weird that myport is not the master port but the node port number. It is like magic that the stub leap some step for you to switch port number for callback function.

Second, two phase commit protocol workflow is important to us that we need to understand completely. As it simple, two phase just master server send request for node server and receive feedback two time, For example, for PUT/DELETE operation, you can see below:

PUT

```
2020-03-29T18:24:29.097: Server 0 is running at port 9876
2020-03-29T18:24:29.102: Server 1 is running at port 8765
2020-03-29T18:24:29.106: Server 2 is running at port 7654
2020-03-29T18:24:29.110: Server 3 is running at port 6543
2020-03-29T18:24:29.113: Server 4 is running at port 5432
2020-03-29T18:24:31.000: Server at port 9876 Received Request: PUT Key:s1 Value:1
2020-03-29T18:24:31.001: Server at port 9876 Send Request to Node Server Ack
2020-03-29T18:24:31.004: Ack received from: 8765
2020-03-29T18:24:31.004: call request for worked. server port: 8765
2020-03-29T18:24:31.007: Ack received from: 7654
2020-03-29T18:24:31.007: call request for worked. server port: 7654
2020-03-29T18:24:31.011: Ack received from: 6543
2020-03-29T18:24:31.013: call request for worked. server port: 6543
2020-03-29T18:24:31.015: Ack received from: 5432
2020-03-29T18:24:31.016: call request for worked. server port: 5432
2020-03-29T18:24:31.116: Wait Ack Try 1
2020-03-29T18:24:31.117: Wait Ack Scuess
2020-03-29T18:24:31.117: Server at port 9876 Send Request to Node Server Commit
2020-03-29T18:24:31.120: Writing the key: s1 and value: 1 at server port: 8765 status:SUCESS
2020-03-29T18:24:31.122: Ack received from: 8765
2020-03-29T18:24:31.122: call commit for worked. server port: 8765
2020-03-29T18:24:31.125: Writing the key: s1 and value: 1 at server port: 7654 status:SUCESS
2020-03-29T18:24:31.126: Ack received from: 7654
2020-03-29T18:24:31.127: call commit for worked. server port: 7654
2020-03-29T18:24:31.129: Writing the key: s1 and value: 1 at server port: 6543 status:SUCESS
2020-03-29T18:24:31.130: Ack received from: 6543
2020-03-29T18:24:31.131: call commit for worked. server port: 6543
2020-03-29T18:24:31.133: Writing the key: s1 and value: 1 at server port: 5432 status:SUCESS
2020-03-29T18:24:31.134: Ack received from: 5432
2020-03-29T18:24:31.135: call commit for worked. server port: 5432
2020-03-29T18:24:31.239: Wait Ack To Commit Try 1
2020-03-29T18:24:31.239: Wait Ack To Commit Scuess
2020-03-29T18:24:31.240: Writing the key: s1 and value: 1 at server port: 9876 status:SUCESS
2020-03-29T18:24:31.240: Server at port 9876 Process: PUT -> key s1 value:1 status : SUCESS
```

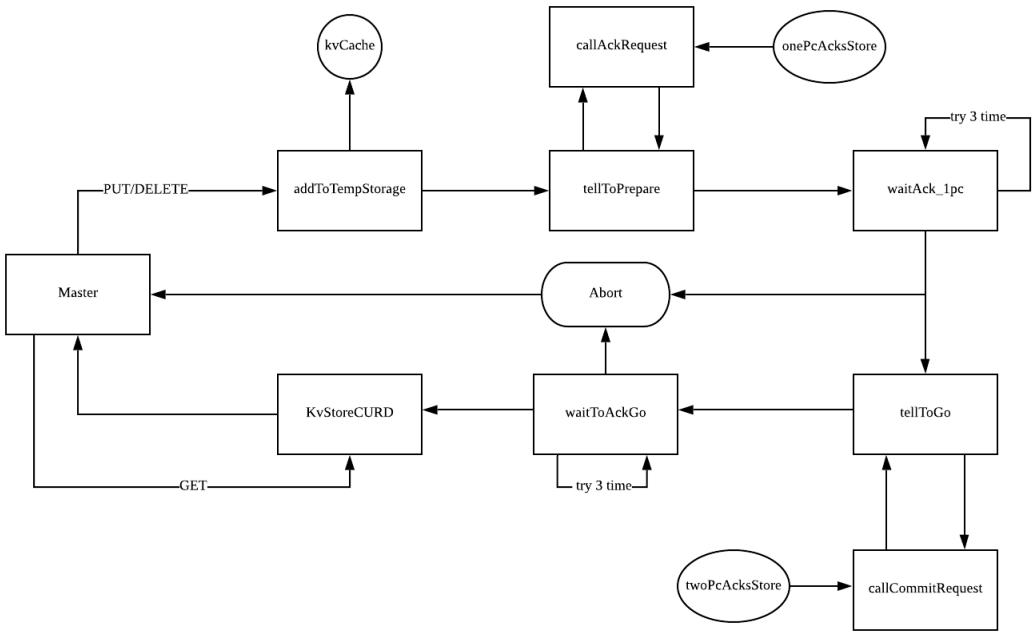
DELETE

```
2020-03-29T18:24:31.724: Server at port 8765 Received Request: DEL Key:s3 Value:
2020-03-29T18:24:31.724: Server at port 8765 Send Request to Node Server Ack
2020-03-29T18:24:31.726: Ack received from: 9876
2020-03-29T18:24:31.726: call request for worked. server port: 9876
2020-03-29T18:24:31.728: Ack received from: 7654
2020-03-29T18:24:31.729: call request for worked. server port: 7654
2020-03-29T18:24:31.731: Ack received from: 6543
2020-03-29T18:24:31.731: call request for worked. server port: 6543
2020-03-29T18:24:31.734: Ack received from: 5432
2020-03-29T18:24:31.734: call request for worked. server port: 5432
2020-03-29T18:24:31.839: Wait Ack Try 1
2020-03-29T18:24:31.839: Wait Ack Scuess
2020-03-29T18:24:31.839: Server at port 8765 Send Request to Node Server Commit
2020-03-29T18:24:31.840: Deleting the key: s3 at server port: 9876 status:FAIL
2020-03-29T18:24:31.842: Ack received from: 9876
2020-03-29T18:24:31.842: call commit for worked. server port: 9876
2020-03-29T18:24:31.844: Deleting the key: s3 at server port: 7654 status:FAIL
2020-03-29T18:24:31.848: Ack received from: 7654
2020-03-29T18:24:31.849: call commit for worked. server port: 7654
2020-03-29T18:24:31.850: Deleting the key: s3 at server port: 6543 status:FAIL
2020-03-29T18:24:31.852: Ack received from: 6543
2020-03-29T18:24:31.853: call commit for worked. server port: 6543
2020-03-29T18:24:31.855: Deleting the key: s3 at server port: 5432 status:FAIL
2020-03-29T18:24:31.857: Ack received from: 5432
2020-03-29T18:24:31.857: call commit for worked. server port: 5432
2020-03-29T18:24:31.962: Wait Ack To Commit Try 1
2020-03-29T18:24:31.962: Wait Ack To Commit Scuess
2020-03-29T18:24:31.963: Deleting the key: s3 at server port: 8765 status:FAIL
2020-03-29T18:24:31.963: Server at port 8765 Process: DELETE -> key s3 status : FAIL
```

server first send Acknowledge to node server, each node server will send back the Ack to master server, then master server start to ack consistency, if all of ack pass which means all node server can prepare to work for commit job. It will go to phase two, server send commit request to each node server, and each node server will start to commit and send the result back to master, if all node search ack passed, the master server will send the result back to client. But if some action is not consistency in ack, the process will be abort and master server will send Fail to client. For GET operation, it only through master server which query the KV store and send result to client.

GET

```
2020-03-29T18:24:31.242: Server at port 8765 Received Request: GET Key:s1
2020-03-29T18:24:31.242: Server at port 8765 Process: GET  key:s1 value:1 Status:SUCESS
```



Third, however, it is not as simple as you think if you combine with RMI or RPC framework in two phase commits implementation. First we need to figure out how to record the state of each request in each node server and master server, I use Random UUID as identifier and use synchronizedMap to save ack state each server. Which is onePcAcksStore and twoPcAcksStore. And there is also kvCache which use in the process of two phase commit. First use tellToPrepare function to trigger callAckRequest to each node server and store ack state. After process request , master start to collect the ack state from each server from onePcAcksStore, there three time can try to collect all Acks except some exception make Abort happen. Succeed from waitAck_1pc function, Master server start to send callCommitRequest to each node server, and each node server start to commit operation with own KvStore and send result to twoPcAcksStore, then in waitToackGo, master collect acks from twoPcAcksStore and only Succeed collection from all Ack can start to operate to master KvStore or throw abort. After Master server get result from its own KvStore or abort, it will send the result to client. That is the whole process of two phase commit in my implementation.