Assignment Overview

Nuozhou Wang

Based on project 3, we have implemented two-phase commit which enables the client-server communication. The server can handle multiple running instances of clients at once with multi-threading, and do operations including PUT, GET and DELETE. All the data among five servers are ensured to be consistent. However, two-phase commit is not fault-tolerance, which means the whole system will down if one or more servers fail. In project 4, it requires us to extends project 3 to be a fault-tolerance system. For my understanding, the purpose and scopes of this project are as follows:

- Update Client class to support the function that client can contact any of the five Key-Value Store server at any time. In project 3, the client looks up remote object with a given server address as the command line argument at the beginning. This server will servers as a coordinator all the time. In project 4, the client should be able to connect one of arbitrary five KV replica servers each time when the client input an operation.

- Implement Paxos: Since that each server has its own HashMap to store the key and values, we want to make sure the data among majority servers are consistent. For the GET operation, we can simply get the value from one replica. However, for the PUT and DELETE operations, we will implement Paxos to guarantees the consensus and fault-tolerance. Paxos is an algorithm that is used to achieve consensus among a set of processes that communicate via an asynchronous network. There are three entities: proposers, acceptors and learners. When received a request from a client, the proposer runs a two-phase protocol with the acceptors. If majority acceptors promise to do the operation, then this operation will be done.

- A “fail” model: To test the whole system is fault-tolerance, we want one or more acceptors to fail at random times. Moreover, the failed acceptor should be able to restart after certain period and should resume the functions of the previous one. In this case, we should modify the servers in project 3 into multi-threaded to make them more controllable.

All in all, this project requires understanding of Paxos and fault-tolerance.

Technical impression

Nuozhou Wang

Detailed implementation will be described as follows:

- Election: Based on the RMI implemented in project 3, to enable the client to connect to arbitrary server's remote object for each request, we will pass server No. 1-5 as a command line argument and the client class will look up the corresponding remote object automatically. When a specific server connects, this server will perform an election to select a proposer as the leader. To simplify this system, we will implement a "fake election" instead of the commonly used bully election and ring election. In our system, the server will try to connect to each proposer to see if they are alive. The server will select the proposer with the largest sequence ID as the leader.

- Paxos: There are several versions of Paxos. Our system implements the initial version. Paxos is a two-phase protocol, in which the proposers interact with the acceptors twice. In the first phase (Prepare-promise), a proposer creates a unique proposal number, ID, and sends a Prepare(ID) messages to acceptors. Each acceptors that receive the message will check the ID in the message and decides if promise. If the ID is larger than the one the acceptor previously have seen, it will store the ID as the max and respond with a promise message. Otherwise, the acceptor will overlook it. If the proposer receives promises from majority of the acceptors, it will proceed. In phase 2 (Propose-accept), the proposer will tell the acceptors to accept the proposal with Propose (ID, value). Each acceptor checks if the ID number is the largest one that it has seen, if will do the operation and reply with an Accepted message, otherwise, it will overlook it. To ensure that the client can continue to send request even if the previous one has no response, we will use the Callable interface and TimeoutException as previously used in project 3. The proposer will also use this method to deal with the situation if acceptors have no response.

- A "fail" model: To achieve the goal that acceptors can fail and restart, we implement the servers with multi-threads. A MasterAndChecker class will initiate the five servers' threads and randomly close one or more servers. This class will also ask and receive the servers' heartbeat to check which one fails. If one server fails, the MasterAndChecker will restart the thread. To ensure that the restarted acceptor will still hold the most updated data, it will send requests to other acceptors and ask for a copy of the stored data.