Assignment Overview

Nuozhou Wang

Based on project 2, we have set up JAVA RMI 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. In project 3, it requires us to extend project 2 from two aspects. 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: In project 2, the client looks up remote object with a given server address. In project 3, single client should be able to connect one of arbitrary five KV replica servers and get consistent data back from the replica server.

- Implement two-phase commit: Since that each server has its own data structure (HashMap) to store the key and values, we need to make sure all the data among five servers are consistent. For the GET operation, we can simply get the value from one replica. However, for the PUT and DELETE operations, we should guarantee that when one request is sent to one server, the other 4 replicas will also do the same operations on their individual sides. We should implement two-phase commit protocol, where the server connected directly with the client is treated as a coordinator, and all the servers are participants.

- Handle timeout cases: Each coordinator should be able to handle unresponsive participants' failure by using some timeout mechanisms. If the coordinator does not receive a response to a particular request, it should be considered as not ready and print out in the coordinator log. With multi-threading implementation, we can assign a new thread and introduce a time-out mechanism.

All in all, this project requires understanding of RMI and two-phase commit. It's a good practice to learn how to implement two-phase commit.

Technical impression

Nuozhou Wang

Detailed implementation will be described as follows:

- RMI: In this project, we should define the remote interface and the remote object. All the methods which will be accessed by client should be stated in the interface. In addition, since that we will use the remote object to implement the two-phase commit, a new remote object is designed to store the key-value pair and do the operations and should be accessed by the coordinator. In this project, all of the five servers will have the same functions, thus we will only create two remote interfaces/objects and bind to the RMI registry by each server with different bind names. Client class will get the object by looking up the class Registry with a String value representing the bind name as a parameter. To enable the client to connect to arbitrary server's remote object, we will pass server No. 1-5 as a command line argument and the client class will look up the corresponding remote object automatically.

- Two-phase commit (2PC): A two-phase commit is a standardized protocol that coordinates all the processes that participate in a distributed atomic transaction. In the first phase, a coordinator makes a request to all the participating processes, asking them whether they are able to commit the transaction. They will return either ready if they can successfully commit the transaction or not ready if they unable to do so. In the second phase, coordinator decides based on the votes whether to send the commit or abort the request. If all the participating processes said ready, then a commit request will be sent to all the processes. If one of them said not ready, then an abort request will be sent to all. In this project, a PUT or DELETE request will be implemented with two-phase commit protocol to ensure the data consistency. Take DELETE operation as an example, the server which is connected by the client directly will act as a coordinator and call the deleteFirstPhaseCommit() function with the key value. This function will connect to ServerQueryRemote object for all the servers. All of the servers will check if the key is stored in their HashMap individually. The key to be deleted will be stored in their local buffers. If they are able to delete the key, they will send "ready" response to the coordinator. The result of these responses will be stored in a Boolean array. After collecting all the responses from the participating processes, the coordinator will decide to commit or abort and call deleteFirstPhaseCommit() to all participates. The participates will delete the key if they receive "commit" message or just empty the buffer. An acknowledgement will be sent back to the coordinator when the participates finish their operations. The final result will be sent to client by the coordinator.

- Callable interface and time-out handling: Callable interface has a single method call() which is meant to include the code that will be executed by the thread. The submit() method will return a special type of value called a Future which can be used to fetch the result of the task when it is available. We will use future.get() method to retrieve the result of the future and add a timeout in the get() method. The future.get() method will throw a TimeoutException if the task is not completed within the specified time. In this project, if the coordinator receives no response from a participate in certain time span, TimeoutException will be thrown, the coordinator will consider the participate not ready.