Project 2: Report

Intro

In this project, I implemented both the client and the server-side code for basic message request/reply operations.

- The KVStore is the application that essentially functions as a key value storage and implements functions to put, get and append from the underlying data structure.
- The AMOApplication is the wrapper around the KVStore which guarantees that the requests from the client get executed at most once (hence the name AMOApplication). The AMOApplication keeps track of the requests that it received and ensures that it does not execute duplicate requests that were triggered by the clients' resend function.
- The SimpleServer is the endpoint for the client to talk to and it is at the highest level of the class hierarchy on the server side. It manages the request from the client and passes it down to the AMOApplication and once it gets a result, it sends it back to the client.
- The SimpleClient is the client which sends the requests and retries if it does not get a reply within a certain timeframe. The retry logic guarantees that the message is sent at least once to the server and so it guarantees that it will execute the command. The client sends command messages to change or retrieve the contents in KVStore.
- Additional classes such as the AMOCommand, AMOResult, Messages, and ClientTimer are used to supplement the workflow.

Flow of Control & Code Design

The SimpleClient is initialized with fields such as the current AMOCommand, the result, and the sequence number. These will be used to keep track of the client's current state. The client sends a request which is a message containing an AMOCommand. This AMOCommand includes the command itself, the sequence number of the message, and the client address. During the sending phase, the current AMOCommand in the client is set to the command that's being sent and the result is set to null. This is to keep track of the state of the client, which is now set to "sent but not received". Once sent, the timer is also set to ensure at least once delivery.

If the message is successfully received by the SimpleServer, it checks to see if the command was previously received. This is done in the AMOApplication which checks to see if the last seen sequence number of the corresponding client is greater than or equal to the sequence number received. This information is stored in clientToSequenceMap, a hashmap of client address to sequence number. Notice that this check is a loose condition and that is acceptable because the client waits for the message reply before sending a new message. In the case that the command is already executed, the server will simply reply to the client with the last sent result for the corresponding client. This ensures the at most once rule. The data structure to store the last seen result is lastSentResultMap, a hashmap of client address to result. If the request was not yet executed, the AMOApplication updates the clientToSequenceMap with the newest sequence number and calls KVStore to execute the command. The server finally replies with the corresponding result from KVStore and updates lastSentResultMap with the latest result.

Once the reply is sent back to the client, it handles the reply by checking to see if the state of the client is "sent but not received" and the reply sent back is the reply the client is waiting for. It then changes the state of the client by updating the current AMOCommand to null, result to the result it received from the server, and incrementing the sequence number by 1 to indicate that the next message sent will be a different message and not a retry message. Then it notifies any threads waiting for the result.

The timer plays a crucial role in guaranteeing at least once delivery. Once the timer runs out, it checks whether the current state of the client is still “sent but not received” and if it is, it sends another request message to the client and resets the timer. This will make sure that the command will be executed and once it confirms that it did execute, the client can progress its state by calling the next command (or terminating).

Design Decisions

In my original design implementation, I considered using randomly generated UUIDs instead of sequence numbers to distinguish messages sent from the client to the server. This was a hacky way to get around the problem of multiple clients starting their sequence number from 0. However, the problem with using UUIDs was that they do not inherently maintain sequential ordering. Although this was not a substantial issue in this project since the client only sends one message before sending a new command message, I anticipated that this design would help in later projects. The cost of using sequence numbers was that the AMOApplication had to distinguish which client’s sequence number was already executed so that we can ensure the at-most-once principle. So, I had to revise the implementation from keeping a set of UUIDs to a hashmap of client to sequence numbers.

Another design decision I made was to add the client address field in the AMOCommand class. This was added due to needing a distinguisher for the AMOApplication to keep track of the client-to-sequence-number map. I considered whether to change the arguments of the execute function in AMOApplication or add the extra client address in the AMOCommand. The trade-off is the message size sent from client to server versus changing the structure of the project. I chose to change the AMOCommand because it seemed unintuitive for the “execute” function to require a client address to execute a command. Upon reflection, if I were to implement a similar project next time, I would perhaps have an additional layer or extra helper function called from the server to the AMOApplication with the client address. Since in the current implementation, the server has the client address so my design is not optimized as the AMOCommand is sending duplication information which would increase the message size, resulting in a possible increase in latency.

Missing Components

Nothing is missing based on the project requirements.

I may be wrong about this or there might be a better design, but as I discussed in the Design Decision section, there could be a feature to pass in the client address to the AMOApplication so that it can use the information to store the sequence numbers and not need to rely on the client address in AMOCommand.

References

I referenced the course repository for the instructions:

https://github.students.cs.ubc.ca/CPSC416-2022W-T2/project2-clientserver