CS 2510

Assignment-1 Report

Dhanush Binu

Introduction

I have implemented the group RPC communication using gRPC and protocol buffers, the server and client implementations are given below. The output is written onto a log file called test_case_1.log.

Server Implementation

In the server code, we have mainly three data structure:

- Connected_list - This list will keep track of all the clients that are currently connected to the server and are participating in the group chat.
- Message_history_dict - This dict will contain a full history of all the previous messages sent by all the clients. This is to ensure that new clients will be able to download the “unread” messages.
- Messages_dict - This dict will keep all the current messages sent by a user. The server will iterate through the recipient list of all the messages and send each one the contents of this message_dict. After all the clients have received the message. The message entry will be dropped from the dict.

In the server class, there are 4 functions apart from the constructor function for the class, these are:

- Connect - This function will take the user_id, user_name as input and will check if the client can access the chat based on whether the client_id of the client is part of the server’s client_ids List. after the client has been authenticated, it will add the client to the connected_client list and send them the “unread” messages. The server will iterate through the Message_history_dict and send all the previous messages sent by other clients.

- Disconnect - This function takes the client_id and client username as input and checks if the client_id is in the connected_client list. If it is then the client_id will be dropped from the connected_clients List.
- Send_message - This function will take the user_name and message as input. It will first authenticate if the user is valid then it will create a new dict with the following structure.

```
message_dict = {
    'sender': user_name,
    'message': user_message,
    'recipients': self.connected_clients.copy()
}
```

This dict contains the sender of the message, the message itself, and the intended recipients of the message(which in this case is the whole list of connected_clients). This dict is then added to the Messages_dict. The message is also appended to the Message_history_dict along with the sender.

- Send_message_to_all - This function only takes the user_name as input. It first iterates through each message in the Messages_dict, it first checks if the user_name is the sender of the message(we don't want the sender of the message to receive their own message) and if the user_name is in the recipients list in the messag_dict. If it is then the function returns the message, after which the user_name is dropped from the recipients list of the message. At the end of function, if the recipients list is empty then the message is dropped from the dict.

Improvements to the server code

I have used a lot of data structures, some of them are unnecessary. Hence those can be removed for more memory efficiency. I have also used two Dictionaries. I feel this could have been implemented through the use of one Dictionary.

Client Implementation

The client has a client_id and the user_name will be provided as an input by the user. The client has 5 functions apart from the constructor function for the class. These are:

- Connect - It will take the user_name and client_id and send it to the server. This function is used to establish a connection to the server and download any unread messages. The response from the server would be a stream of messages. The function will take these messages and print them out, along with the sender.
- Send_Message - This function takes the input from the stdout and sends the message to the server function. The server will return an empty object as response.
- Receive_message Function - This function will take the user_name as input and check if any other client has sent any message and will fetch and display it to the client. The server function will return a stream of messages and all that will be printed to the client terminal.
- Input_thread Function - This function will take the user_input and invoke the Send_message function by passing the client_username and message as parameters.
- Start Function - This function will first check if the client is connected to the server or not , if it is then it will start two threads that will concurrently execute the receive_messages and input_thread function.

Client Code Improvements

I have created two threads, which simultaneously take the input message and receive messages from the server. But the receive_messages thread has a while loop that continuously sends requests to the server, asking for any new messages. I think there might be better ways to do this. Could make the Receive_messages thread run at an interval.