Group Chat System - Distributed RPC Communication

Prepared by - Dikshya Pahari
Course: 2510 Computer Operating System
Professor : Dr. Stephen Lee Date: 02/15/2025

1. Introduction

This report documents the design, implementation, and testing of a **distributed group chat system** using **gRPC** in Python. The system follows a **client-server architecture**, where multiple clients communicate asynchronously through a central server. The goal was to implement a scalable and efficient message-passing system while ensuring **message delivery tracking** and handling **unread messages** properly.

2. Design Choices & Implementation

2.1 Architecture

- The system consists of **one server** and **multiple clients**.
- Clients communicate **asynchronously** with the server using **gRPC**.
- The server **stores messages** and ensures that each client receives only **unread messages**.

2.2 Components

1. `server.py` - Handles message storage, client registration, and message retrieval.
2. `client.py` - Allows clients to send and receive messages asynchronously.
3. `chat.proto` - Defines the **gRPC service**, including message formats and remote procedure calls.
4. `run_test_1.py` - Automates testing by simulating different client scenarios.
5. `Dockerfile` - Packages the application into a container for easy deployment and execution.

2.3 Key Features

Asynchronous RPC Communication - Clients and server interact without blocking operations.
Unread Message Handling - The server tracks which messages each client has received.
Multi-Client Support - Any number of clients can join and communicate.
Docker Deployment - The system can be easily built and executed using Docker.

3. Testing and Results

3.1 Test Cases

The following scenarios were tested using `run_test_1.py`:

1. **Alice sends a message before Bob and Chad come online.**
    - Expected: Bob and Chad receive Alice's message once they connect.
2. **All clients are online, and Bob sends a message.**
    - Expected: Alice and Chad receive the message, but Bob does not.
3. **Doug joins the server but is not part of the group.**
    - Expected: Doug does not receive any messages.

3.2 Execution Logs

Sample output from the test runs:

🔹 Starting test scenario…
Alice sent: "Hello everyone!"
Bob and Chad joined and received: "Hello everyone!" from Alice
Bob sent: "Hi Alice and Chad!"
Alice and Chad received: "Hi Alice and Chad!" from Bob
Doug joined but received no messages
Test completed successfully!

4. Possible Improvements

- **Message Persistence**: Storing messages in a database instead of memory would prevent data loss.
- **User Authentication**: Adding authentication tokens for secure access.
- **Group Customization**: Allow clients to create different chat rooms dynamically.
- **Better Logging**: Implement structured logging for debugging and monitoring.

5. Conclusion

This project successfully implemented a **distributed RPC-based chat system** that efficiently handles **asynchronous communication using gRPC**. The system is fully **containerized with Docker**, ensuring **easy deployment and testing**. The design choices focused on **scalability and efficiency**, making it a solid foundation for future enhancements.