Q-4 (HW-4)

Real Time Document

Report

Akshit Sharma
2021101029

This tool has been implemented using GRPC for client-server communication. Golang has been used for both the clients and server code. There are 2 types of clients: a logger client that logs the changes and the client that has made those changes to the live document, and a user client who can view and edit the live document using a web client, through which communication occurs using web sockets.

Server Implementation

The server is implemented using gRPC and maintains the current state of the document. It handles multiple clients and synchronizes updates between them.

Key features:

- Uses a mutex lock for safe access to shared resources
- Maintains the stream information of connected clients
- Handles document updates, stores its latest knowledge of the document contents and broadcasts them to other clients
- Implements logging functionality

This implementation meets the following requirements:

- **Real-time collaborative editing:** The server receives updates from clients and broadcasts them to others, so that they update their version of the shared document to maintain consistency.
- **Support for multiple clients:** The server maintains a map of connected clients.
- **Logging of document changes:** The server logs all updates received from clients, by sending the log request to a separate client (called logger).

Client Implementation

The client acts as a bridge between the web client and the gRPC server. It maintains **WebSocket** connections with browsers and translates WebSocket messages to gRPC calls and vice versa.

Key features:

- Establishes gRPC connection with the server
- Handles WebSocket connections from browsers
- Uses Bidirectional Streaming for its communication with server
- Manages the local state of the document

This implementation meets the following requirements:

- **Real-time collaborative editing**: The client sends updates to the server and receives updates from server, for the updates made by the other clients.
- **Live Document interface on Web**: Each client instance can handle a WebSocket connection from a browser, to display the current version of the live document.

Web Client:

The web client provides a simple text-area for users to edit the document and uses WebSocket to communicate changes to the gRPC client.

Key features:

- Sends updates to the server when the document is modified (handled using JS code), on the go, without any submit/save button
- Updates the text-area contents when receives changes from the client

Logger Implementation

The server sends request to a special client, called the logger, for all changes that are made by a client, and sent to the server. It keeps appending these logs in a text file called **logs.txt.** The server is notified by it, once it is started and the server

maintains this connection till the logger runs. It uses a single direction streaming RPC to communicate with the logger.

The Live document implementation, in simple words

The live document implementation enables real-time collaborative editing by connecting multiple clients to a gRPC server, which maintains the current state of the document and synchronizes updates between clients. Each client communicates with the server via bidirectional streaming, sending document changes and receiving updates from other clients to ensure consistency, through the server. The web client provides a user-friendly interface with a text-area where users can edit the document, and changes are immediately communicated via WebSocket to the gRPC client, which relays them to the server. The server logs all changes by communicating with a dedicated logger client, which appends these updates to a log file. This architecture supports seamless real-time collaboration, handling multiple users while maintaining a consistent, up-to-date shared document.

So, this implementation meets the requirements of the real time document that we are required to implement. Client sends changes to server. There is one client, which sends changes to the single server (connected to all clients), which in turn sends changes to a logging program. These multiple clients are thus synced using the server, without having any communication among the clients.

logs.txt

Q4 > client > logs.txt

```
2024-10-14T23:04:54+05:30 Client: client_8081 updated doc - Change: a Type: add Position: 0
2024-10-14T23:31:35+05:30 Client: client_8080 updated doc - Change: a Type: add Position: 0
2024-10-14T23:31:56+05:30 Client: client_8080 updated doc - Change: k Type: add Position: 1
2024-10-14T23:31:56+05:30 Client: client_8080 updated doc - Change: s Type: add Position: 2
2024-10-14T23:31:56+05:30 Client: client_8080 updated doc - Change: h Type: add Position: 3
2024-10-14T23:31:56+05:30 Client: client_8080 updated doc - Change: i Type: add Position: 4
2024-10-14T23:31:56+05:30 Client: client_8080 updated doc - Change: t Type: add Position: 5
2024-10-14T23:31:57+05:30 Client: client_8080 updated doc - Change:   Type: add Position: 6
```