Distributed Systems HW-4 Q4 Report

Team Number: 55

Team Members: Mitansh Kayathwal, Pradeep Mishra

Roll Numbers: 2021101026, 2023801013

Live Document Editor System (Approach Taken)

Overview

The Live Document Editor system consists of three main components:

1. Client (Web Application)
2. Server (gRPC Server)
3. Logger (Separate Client for Logging)

These components work together to provide a collaborative document editing experience where multiple users can edit a document simultaneously, with changes synced in real-time across all connected clients.

1. Proto File Explanation (Messages & RPCs)

The message `DocumentChange` represents a change made to the document. It contains the following fields:

1. `client_id` (string): Identifies the client that made the change.
2. `content` (string): The actual content of the change.
3. `position` (int32): The position in the document where the change occurred.
4. `change_type` (string): Indicates the type of change ("add", "delete", "edit").
5. `timestamp` (string): The time when the change was made.

`CollaborativeDocumentService` defines 2 Remote Procedure Calls:

- `SyncDocumentChanges`:
  - Input: stream of DocumentChange
  - Output: stream of DocumentChange
  - This bidirectional streaming RPC allows clients to send and receive document changes in real-time. Clients can continuously send changes to the server and receive changes from other clients.

- `StreamDocumentLogs`:
    - Input: EmptyMessage
    - Output: stream of DocumentChange
    - This server-streaming RPC allows a client (typically the logger) to receive a stream of all document changes. It takes an empty message as input because it doesn't require any initial parameters.

This service containing the 2 Remote Procedure Calls helps us to implement our Live Document Editing system.

2. Client (Web Application)

The client is a web application that provides the user interface for document editing. It's built using HTML, JavaScript, and WebSocket for real-time communication.

Key Features:

- **WebSocket Connection**: Establishes a WebSocket connection to the server for real-time updates.
- **Text Editor**: A text-area like element that allows users to edit the document.
- **Change Detection**: Monitors user input and detects changes in the document.
- **Change Transmission**: Sends detected changes to the server via WebSocket.
- **Remote Update Handling**: Processes updates received from the server and applies them to the local document.

Detailed Functionality:

1. **WebSocket Setup**:
    - On page load, a WebSocket connection is established to the server.
    - The connection status is displayed to the user live on the webpage.
2. **Editor Initialization**:
    - The textarea is populated with the initial document content received from the server. This will contain the current state of the document of clients who have worked on the document, or will be empty in case there have not been any edits till now.
3. **Change Detection**:
    - The last known content and cursor position is tracked.
    - On input events, I compares the current content with the last known content to determine the type of change (add, delete, or replace).
4. **Change Transmission**:
    - When a change is detected, it's formatted into a JSON object containing:
        - `change_type`: "add", "delete", or "replace"

    - `addContent`: The content added (if any)
    - `addPosition`: The position where content was added
    - `deleteContent`: The content deleted (if any)
    - `deletePosition`: The position where content was deleted
  - This change object is sent to the server via WebSocket.

5. **Remote Update Handling**:
   - When a message is received from the server, it's parsed as JSON.
   - The local document content is updated with the received changes.
   - The cursor position is adjusted to maintain its relative position in the document. (Additional functionality)

6. **Conflict Resolution**:
   - The system uses an optimistic concurrency model.
   - Local changes are applied immediately and sent to the server.
   - Remote changes are applied as they are received, overwriting local changes if they affect the same region.

3. Server (gRPC Server)

The server is implemented in Go and uses gRPC for communication with clients. It manages the document state and coordinates changes between clients.

Key Features:

- **gRPC Service**: Implements the `CollaborativeDocumentService` defined in the protocol buffer.
- **Document State Management**: Maintains the current state of the document.
- **Client Management**: Keeps track of connected clients and their streams.
- **Change Broadcasting**: Distributes changes to all connected clients.
- **Logger Integration**: Sends all changes to a connected logger client.

Detailed Functionality:

1. **Server Initialization**:
   - The server starts listening on a specified port.
   - It initializes an empty document and a map to store client connections.

2. **Client Connection Handling**:
   - When a client connects, a new gRPC stream is established.
   - The client is added to the list of connected clients.

3. **Document Synchronization**:
   - When a client first connects, it receives the current state of the document.

4. **Change Processing**:
    - The server receives changes from clients through the gRPC stream.
    - Each change is processed and applied to the server's document state.
    - The change is then broadcast to all other connected clients.
5. **Concurrency Handling**:
    - The server uses a mutex to ensure thread-safe access to the document state and client list.
    - Changes are processed sequentially to maintain consistency.
6. **Logger Integration**:
    - If a logger client is connected, all changes are also sent to the logger stream.
7. **Client Disconnection**:
    - When a client disconnects, it's removed from the list of connected clients.

4. Logger (Separate Client for Logging task)

The logger is a separate client that connects to the server and receives all document changes, writing them to a log file.

Key Features:

- **gRPC Client**: Connects to the server as a special logging client.
- **Log File Management**: Opens and manages a log file for writing.
- **Change Logging**: Receives all document changes and writes them to the log file in a formatted manner.

Detailed Functionality:

1. **Logger Initialization**:
    - The logger opens a log file for writing.
    - It establishes a gRPC connection to the server.
2. **Log Streaming**:
    - The logger starts a stream to receive document changes from the server.
3. **Log Processing**:
    - For each received change, the logger:
        - Parses the timestamp
        - Formats the log entry based on the change type
        - Writes the formatted log entry to the log file
4. **Continuous Operation**:

- The logger continues to receive and log changes until the connection is closed or an error occurs.

System Integration and Flow

1. **Startup**:
    - The server is started first, listening for client connections.
    - The logger client connects to the server.
    - Web clients connect to the server through their browsers.
2. **Editing Flow**:
    - A user makes a change in their web client.
    - The change is detected by the client-side JavaScript.
    - The change is sent to the server via WebSocket.
    - The server receives the change and:
        - Applies it to the central document state
        - Broadcasts it to all other connected clients
        - Sends it to the logger client
    - Other clients receive the change and apply it to their local document state.
    - The logger receives the change and writes it to the log file.
3. **Conflict Resolution**:
    - If two clients edit the same part of the document simultaneously:
        - Both changes are sent to the server.
        - The server processes them in the order received.
        - All clients receive both changes and apply them sequentially.
    - This may result in the last change "winning", but all clients will end up with the same final state.

Conclusion

This system achieves live document editing through:

1. Real-time communication using WebSockets and gRPC streams.
2. Centralized document state management on the server.
3. Immediate local updates combined with server-based synchronization.
4. Broadcasting of changes to all connected clients.
5. Sequential processing of changes to maintain consistency.

The combination of these techniques allows multiple users to edit the document simultaneously, with changes reflected across all clients in near real-time, creating a collaborative editing experience. Error handling is done by multiple retries to the server in case of disconnection.