CS4459 Group Project Report

Eric Liu - 251071062
Jiahao Tang - 251175958
Adrian Koziskie - 251163461
Modin Wang - 251110972

The Problem:

The problem that we tackled was live collaboration on a text document. The ability for multiple users to simultaneously edit a document streamlines the writing process. Implementing a distributed system to solve this problem entails ensuring a consistent state of the document for all users and efficiently passing and storing data in potentially large documents. Challenges we overcame included issues related to concurrency and ensuring edits didn't lead to conflict or data loss on the state of the document.

Our Project:

We used gRPC and WebSockets to create a real-time collaborative text editor. Additionally, we worked on optimizing the editor's responsiveness and performance by implementing a piece table data structure to update the document in the database efficiently and to reduce the size of messages between the client and server.

Our Design:

We began the design stage by drafting our proto file to contain request messages for actions that would be taken in a collaborative text editor: insert requests, delete requests, and a fetch request for the state of the document. The system also needed response messages to make sure the server and client were acknowledging an edit.

Early on, we selected a data structure called a piece table to represent the changing state of the document. A piece table consists of two main components: the original document and a buffer for all additions (inserts). In our proto file, we represented the piece table through the function ***message Piece*** which contained the buffer indicating whether the piece is from the 'original' or 'add' buffer. We then represented the serialized state of the document through another function called ***message SerializedDocumentState*** that contained the 'original' buffer and 'add' buffer as well as the list of pieces that made up the document.

After compiling our proto file, we created server and client python files. Within our server architecture, functions defined in the proto file are implemented for document state initialization and manipulation via the piece table structure. These enable efficient Insert, Delete, and Fetch operations for real-time text editing. This approach ensures optimal performance and seamless text manipulation, leveraging the strengths of the piece table for minimal data movement and instant access to document segments. The server file also connects to the MongoDB server used to store the document on the server side. Additionally, the server code saves backup snapshots of the document on a schedule (using APScheduler) into the database.

To enable real-time collaborative editing, our architecture integrates WebSocket technology. This provided a low-latency connection between the client and server, facilitating the instantaneous broadcast of document changes to all connected clients. Upon any edit action, the change is immediately transmitted over the WebSocket connection. The server then broadcasts this update to all clients, ensuring that each user's view of the document is synchronized in real-time. The Socket.IO library abstracts WebSocket connections to offer compatibility across a wide range of browsers and fallback options for environments where WebSockets are not supported. This contributed to achieving the design goal of a highly responsive editing environment.

To craft the user interface, we utilized HTML for its structure, CSS for its styling, and JavaScript for dynamic integration, weaving together scripts and styles to bring the editor's functionality to life.

We established a central code repository on BitBucket, utilizing Git for version control to systematically manage and maintain the project's codebase. The codebase includes a requirement.txt file to facilitate quick installation of dependencies within a python virtual environment. This allowed team members to get the system up and running on their system quickly.

Challenges:

In the early stages of our project, we encountered a formidable challenge: the integration of JavaScript with gRPC. This hurdle initially seemed insurmountable, primarily because compiling the .proto file with JavaScript was proving to be a complex task. Our breakthrough came with the discovery of gRPC Web, a JavaScript adaptation of gRPC designed specifically for browser clients. This tool became our bridge, enabling the compilation of the essential .proto files. More than that, it facilitated a seamless connection to gRPC services through the use of a specialized proxy powered by Envoy. This solution not only overcame our initial obstacle but also paved the way for a more robust and efficient communication framework within our application.

Another challenge we faced during the development of our project was the lack of familiarity with MongoDB within our team. MongoDB, a leading NoSQL database, is instrumental in handling data storage and retrieval needs for the application. To address this challenge, we began by identifying useful learning resources like online tutorials and MongoDB documentation. We held regular meetings where team members would present their findings and solutions to encountered problems. After putting in some effort, we successfully got the MongoDB server to run and integrated it with our application. This experience has equipped us

with valuable new technical skills and strengthened our resolve to tackle future development challenges with confidence and a collaborative spirit.

As the list of dependencies for the project grew, it became increasingly challenging to get the system running on everyone's setup. We faced issues with connecting to the bitbucket repository and with setting up MongoDB on different operating systems. Teamwork was instrumental in overcoming these issues; team members who had the system working shared their experiences and focused on resolving the problems. This ensured that nobody was left behind and allowed development to proceed swiftly.