Real Time Collaboration Platform

Final Project in Distributed Systems Architectures

Ignazio Emanuele Piccichè
Mattia Castiello
Michela Di Simone

Academic Year 2023/2024

Indice

1 Introduction

The project aims to develop a real-time collaboration platform that not only supports shared document editing but is also resilient to network partitions, thereby ensuring a reliable user experience.

1.1 Project Objectives

The main objective of the project is to create a collaborative editor capable of:

- Allowing multiple users to edit the same document simultaneously, leveraging the use of WebSockets
- Maintaining data consistency and user modifications in the presence of conflicts, using the *Conflict-free Replicated Data Types* (CRDTs) technique
- Handling network partitions to ensure that all changes are correctly integrated once the partition is resolved
- Effectively notifying the presence of users, informing who is currently connected to the shared document

1.2 Challenges

To verify the robustness of the platform, targeted experiments were conducted including:

- The simulation of network partitions to test the system's ability to reconcile changes
- The analysis of the impact of different conflict resolution strategies on the consistency and latency of the system

In conclusion, the project will not only explore the complexity of real-time collaboration but will also seek to provide practical and innovative solutions to ensure the continuity and integrity of shared work in variable network scenarios.

2 Back-End

Within *FileServer.py*, which has been developed, real-time collaboration on a shared document is managed via *WebSocket,* allowing multiple clients to connect and interact simultaneously with a file, updating it and maintaining consistency between versions across different clients.

The application uses a CRDT (*Conflict-free Replicated Data Type*) algorithm to resolve conflicts and ensure that all clients view the same updated version of the document.

2.1 Connection Management

This section describes how the server manages WebSocket connections, keeping a log of connected clients and updating other clients when a new user connects or disconnects.

2.1.1 Method *file_server*

- This method handles connection and communication with clients. Upon connection initiation, the server requests the client's username and adds it to the list of connected users. In case of disconnection, the server removes the client from the list and informs other users. Additionally, it handles CRDT operations and maintains document consistency.

2.2 Message Management

This section explains how the server manages and transmits changes between clients. This includes both document update messages and system status messages.

2.2.1 Method *send_to_all*

- This method sends a message to all connected clients. It is used to notify all users about events such as user entry or exit.

2.2.2 Method *broadcast*

- Similar to *send_to_all,* but specifically designed to send file content updates to all clients. This method uses JSON to serialize messages, facilitating the sending of structured data.

2.2.3 Method *send_user_list*

- This method updates all clients with the current list of connected users, maintaining visibility on participants.

2.3 Document Management

This section explores how the server loads, saves, and updates the content of the shared file.

2.3.1 Method *load_file*

- Loads the content of the shared file from the file system, if it exists. If the file does not exist, it returns an empty string.

2.3.2 Method *save_file*

- Saves the content of the shared file to the file system, updating the file with recent changes.

2.3.3 Method *network_partition_consistency*

- Upon user reconnection, resolves conflicts between file versions obtained from different sources (users), using the *SequenceMatcher* method from the *difflib* library to align the changes.

2.4 CRDT Operations Management

This section addresses how the server uses CRDT to apply and synchronize document changes.

2.4.1 Method *crdt_operations*

- Calculates the necessary operations to transform an old text into a new one, generating a list of operations representing additions, deletions, or modifications. These operations are then applied to the CRDT class to maintain consistency among clients.

3 Front-End

Within the file ***style.css***, the visual style and layout of the web page are defined.
Within the file ***HomePage.html***, the structure of the web page is defined, allowing users to enter their username, edit a document in real time, and observe the status and list of online users. Additionally, it manages the logic for real-time communication and user interaction. The key functionalities implemented are:

- **User Management**: Users can enter a username and connect to the server.
- **Document Editing**: Users can type content into a shared text area.
- **Real-time Collaboration**: Changes made by one user are reflected in the document for all connected users.
- **Cursor Position Synchronization**: The cursor position of each user is monitored.

3.1 WebSocket Connection

The *connectedWebSocket* function establishes a WebSocket connection with the server on port 5555:

- **Onopen**: When the connection is opened, any local changes are sent to the server.
- **Onmessage**: Handles messages received from the server, updating the text file content and the list of online users.

3.2 JSON Validation

The *isValidJSON* function checks if a string can be converted to a JSON object, returning *true* if it can, otherwise returning *false*.

3.3 Timestamp Management

The *toggleTimestamps.onchange* function manages the display of timestamps in status messages based on the checkbox state.

3.4 Setting Username

The *setUsernameButton.onclick* function handles the click event on the *Set Username* button, sending the username to the server, enabling the editor if the username is not empty, and providing the option to disconnect.

3.5 Editor Change Management

The *editor.addEventListener('input')* function detects changes in the editor, creates a message object containing the updated content and username. It stores the message locally and sends it to the server, captures the current position in the editor, and stores it in a variable.

3.6 Cursor Position Management

3.6.1 Sending Cursor Position

The *sendCursorPosition* function retrieves the current cursor position from the editor and sends the username and cursor position to the *setCursorPosition* server-side function.

3.6.2 Updating Cursor Position

The *setCursorPosition(position)* function updates the document state and cursor position for that user.

3.7 Partition Simulation

The *simulatePartitionButton.onclick* function simulates a network partition by closing the WebSocket connection. It then enables the reconnect button and disables the *Simulate Partition* button. Additionally, when the user disconnects, the status and user list displays are cleared.

3.8 Reconnecting to the Server

The *reconnectButton.onclick* function handles the click event on the *Reconnect* button, calling the *connectedWebSocket* function to reestablish the connection to the server.

4 Docker

The project utilized *Docker* and *Docker Compose*. Docker is an open-source platform that automates the deployment of applications within software containers. Containers are lightweight, portable, and self-sufficient, meaning they include everything needed to run the application: code, system libraries, and settings.

The decision to adopt Docker in this project was driven by the following reasons:

- **Consistent Environments**: Ensure that the application behaves identically during development, testing, and production
- **Ease of Deployment**: Simplify the deployment of the application across different environments without having to reconfigure the system
- **Scalability**: Facilitate the scaling of services, allowing for more efficient handling of load peaks
- **Isolation**: Separate different components of the application, improving security and dependency management

Below are the configuration files used.

4.1 Dockerfile for Nginx

The Dockerfile for Nginx is used to configure a container that serves static web content. The content of the Dockerfile is as follows:

```
# Use the official Nginx image from the Docker Hub
FROM nginx:alpine

# Copy the custom Nginx configuration file into the container
COPY HomePage.html /usr/share/nginx/html/index.html
COPY style.css /usr/share/nginx/html/style.css
```

Listing 1: Dockerfile for Nginx

4.2 Dockerfile for the Python Server

The Dockerfile for the Python server configures a container to run a Python application. The content of the Dockerfile is as follows:

```
# Use the official Python image from the Docker Hub
FROM python:3.10

# Set the working directory in the container to /app
WORKDIR /app

# Copy the current directory contents into the
   container at /app
COPY . /app

# Install the dependencies
RUN pip install -r requirements.txt

# Run app.py when the container launches
ENTRYPOINT ["python", "FileServer.py"]
```

Listing 2: Dockerfile for the Python Server

4.3 Docker-compose.yaml File

The ***docker-compose.yaml*** file was used to orchestrate the two services defined in the Dockerfiles above. The content is as follows:

```
version: '3.8' # version of docker-compose

services: # services that will be run
  web:
    build:
      context: ./web
    ports:
       - "8583:80"
    networks:
      - sharedDockNet

  server:
    build: # build the image from the current directory
      context: ./server
    ports:
      - "5555:6000"
    networks:
      - sharedDockNet

networks: # network named
  sharedDockNet:
```

Listing 3: Docker-compose

5 Compute Engine (GCP)

The program has been implemented within a *Compute Engine* from *Google Cloud Platform,* using a small-sized virtual machine with the following characteristics:

- **Machine Type**: e2-micro
- **CPU Platform**: Intel Broadwell
- **Architecture**: x86/64
- **GPU**: None

The choice of an e2-micro machine was driven by the need to keep operational costs low while ensuring adequate performance to handle real-time collaboration on shared documents. The Intel Broadwell CPU platform provides a good balance of energy efficiency and computing power, sufficient for our needs.

5.1 Firewall Configuration

To ensure the proper functioning of the service, it was necessary to configure the firewall rules in Google Cloud Platform's *VPC Network* service. This included enabling the following inbound ports to allow communication between clients and the server:

- **TCP Port 5555**: Used for the server (backend) service. This port is crucial for managing CRDT operations and synchronizing documents between clients.
- **TCP Port 8583**: Used for the web service (client). This port allows users to access the web interface to modify documents.

The IP ranges have been configured to `0.0.0.0/0`, allowing access from any IP address. This was necessary to ensure that clients can connect to the server regardless of their geographical location, facilitating global collaboration without restrictions.

5.2 Service Access

The site is accessible at the following address: `http://34.154.106.176:8583/`

This configuration allows users to access the web service for collaborative real-time document editing.

Thanks to Google Cloud's scalable infrastructure, the program can handle a growing number of users without compromising performance. This ensures an efficient and uninterrupted collaboration experience.

6 Future Developments

Although the current project represents a solid foundation for a shared collaboration platform, there are some areas for improvement and potential advanced features that could be developed in the future to enhance user experience and system robustness:

1. **User Change Tracking**

   Currently, each online user is allowed to edit the document without tracking who made a particular change. A possible improvement could be the implementation of a system that tracks changes made by each user and allows:

   - Viewing the cursor of each user when they are editing text in real time
   - Using a specific color for each user to highlight the changes that user has made

2. **Ability to Download the Document**

   Another enhancement could be the ability to download the document in various formats (PDF, docx, txt), allowing offline access and tracking of different versions of the document.

3. **Merge Conflict Resolution**

   When a user experiences a network partition and then reconnects, the offline text merges with what has been written by online users. However, bugs can sometimes occur. Therefore, it is essential to:

   - Implement more robust merge algorithms that can better handle conflicts between words
   - Provide the ability for notification and manual resolution of conflicts in more complex cases

4. **Stylish Text**

   To make the editor more versatile, it would be appropriate to add text customization features. This would include:

   - Basic formatting options such as bold, italic, underline, text color, font size
   - Integration of advanced formatting options such as bullet points, numbering, inserting links, images