The Fu Foundation School of Engineering and Applied Science

Topics in Network - Cloud Computing and Virtual Network

(ELEN 6770)

Project Report

Demo for Real-time Collaborative Editor

Instructor:	Prof. Krishan Sabnani	krishan.sabnani@nokia-bell-labs.com
Instructor:	Prof. Thomas Woo	thomas.woo@nokia-bell-labs.com
Course Assistant:	Xiaoyang Liu	xl2427@columbia.edu

Name	Student ID	Contact Information
Changxu Luo	cl3875	changxu.luo@columbia.edu

Table of Contents

1 Overview

1.1 Introduction to Real-time Collaborative Text Editor

1.1.1 What is a Text Editor

A traditional text editor is a space where people can insert or delete text characters and then save the resulting text to a file. This definition can also be applied to Rich-text editor since attributes such as font or size can be represented as a special combination of characters in the raw text, and then be represented as the rich-text format through the editor rendering.

For a traditional text editor, the two basic operations, insert and delete, are combined with the character value and position. Consider that in the beginning, a text editor has the text of "LOUD", with each character position as shown in the following table:

Character	L	O	U	D
Position	0	1	2	3

Then suppose now a user wants to insert character "C" in the beginning, the operation can be represented as "insert("C", 0)", and the table would become:

Character	C	L	O	U	D
Position	0	1	2	3	4

Then if the user wants to delete character "O" at position 2, the operation can be represented as "delete(2)", and finally the table would become:

Character	C	L	U	D
Position	0	1	2	3

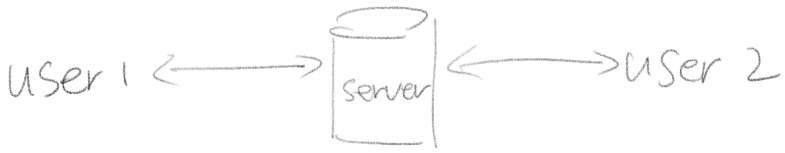
1.1.2 What is a Collaborative Text Editor

In traditional text editor, since they are run locally, they can only be used by one user at one place at one time. With the development of the Internet and cloud services, people want to have a way for editing the groups producing works together. A good example of such a collaborative editor is Wikipedia where different people can contribute to the same page. Another example of collaborative editor can be a version control system like Git.

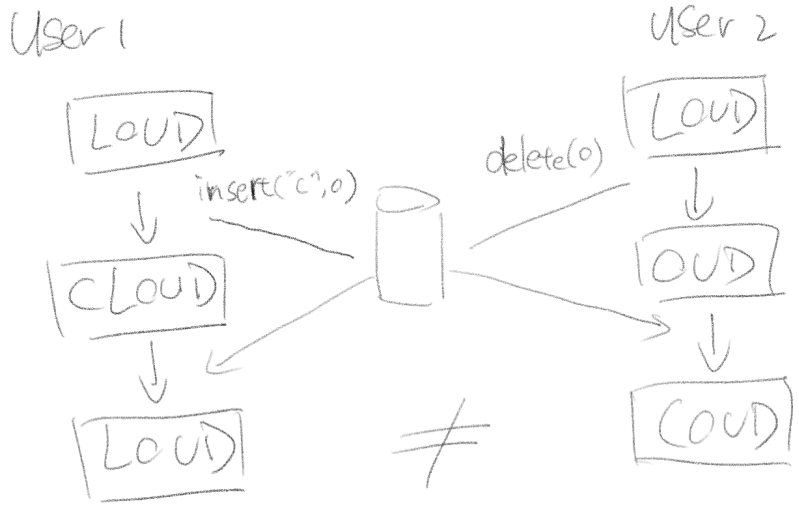
However, two examples above can only be called the non-real-time collaborative editors. They don't allow the users to simultaneously editing the same document. For example, when two collaborator are editing the same document by Git, they won't be able to see others' edits at the same time, and when they push the file to the repository, it may cause conflicts and require users to manually merge the differences.

1.1.3 Real-time Collaborative Text Editor and Its Difficulty

With the need of real-time collaborative editing, many real-time collaborative editing services have emerged these years. Some famous examples include Google Docs and Overleaf. Here a basic structure with a relay server will be used to introduce the problems people encounter when developing a real-time collaborative text editor. Consider user 1 and user 2 are editing one document together. Their operations will both be sent to a central relay server and then sent to the other users. The structure looks like this



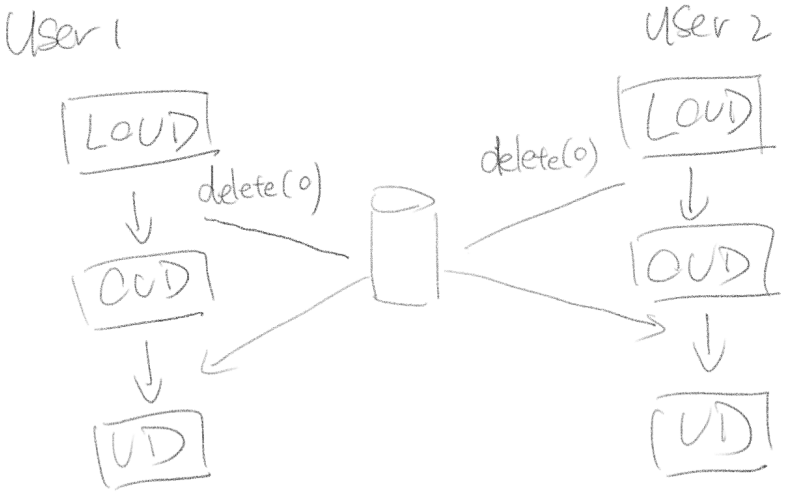
Now suppose that initially, the editor has content "LOUD". And then at about the same time, user1 wants to insert C at position 0 (insert("C", 0)), while user2 wants to delete the first character. The procedure is described in the following figure:



We can see that, when both users are informed about the operation done by the other user, the resulting text are not the same. This means that their documents does not converge to the same state. Actually, to keep the editors converging to the same state is one of the primary challenges to build a real-time collaborative editor. The reason why user's texts in the example above don't converge is that the operations of insert and delete are not commutative, which means that different order of executing these commands can result in different output.

Another problem for the real-time collaborative editor is when different users executing the same oper-

ations as shown in the following figure:



In this example, two users want to delete character at the position 1 simultaneously. The results converge as both sides get the same text, however, what both users want is the result of "OUD" instead of "UD". This means that the delete operations are not idempotent, which means that the repeated delete operations can cause different result. In a real-time collaborative editor, we have to guarantee that the delete operation is idempotent, otherwise users will have to remember the value and position of the mis-deleted character in order to insert it back to the document. According to Conclave Team, "for a collaborative text editor, the insert and delete operations mush commute and the delete operations must be idempotent"[1].

1.2 Algorithms for Real-time Collaborative Editing

1.2.1 Operational Transformation (OT)

Operational Transformation (OT) is one possible algorithm that can be used to solve the problems of commutativity and idempotency. Basically, OT is an algorithm to transform the parameters of an editing operation according to the effects of previously executed concurrent operations so that the transformed operation can achieve the correct effect and maintain document consistency. Some important examples of services that apply OT include Apache Wave and Google Docs.

With all the saying, OT is a very difficult algorithm and there are many different OT systems. Here only the basic structure of an OT system would be presented. The established strategy of designing OT systems is to separate the high-level transformation control algorithms from the low-level transformation functions as shown in the following table:

[1] https://conclave-team.github.io/conclave-site/#conflict-free-replicated-data-type-crdt

OT Control ALgorithm
determine which operations are transformed against others according to their concurrency/context relations

OT Properties and Conditions
divide responsibilities between algorithms and functions

OT Transformation Functions
determine how to transform a pair of primitive operations according to operation types, positions, and other parameters

1.2.2 Conflict-free Replicated Data Types (CRDT)

Since OT is very complicated, researchers want to strengthen and simplify OT. OT is implemented without changing the fundamental structure of a basic editor, which treats each character as having a value and a position. However, this result in difficulty in solutions for keeping the operations to be commutative. With this thinking, a data structure called Conflict-Free Replicated Data Type (CRDT) was introduced. There are two approaches to CRDTs, both of which can provide strong eventual consistency: operation-based CRDTs and state-based CRDTs[2]. CRDTs take a different approach, which doesn't treat each character as just having a value and absolute position. Instead, properties are added to each character object that enabled commutativity and idempotency. This results in a much simpler algorithm than OT. Some critical requirements for CRDTs include globally unique characters and globally ordered characters.

1.3 Yjs CRDT Algorithm

In this project, Yjs is applied as the library of CRDT algorithm. Yjs is a CRDT implementation that exposes its internal data structure as shared types. It is network agnostic, supporting many exisitng rich text editors, offline editing, version snapshots, undo/redo and shared cursors. It scales well with an unlimited number of users and is well suted for even large documents. The information of Yjs can be found at **here**.

Yjs implements a modified version of algorithm described in paper "Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types"[3]. Here are the examples for what its algorithm does[4]:

1. If a user inserts elements in sequence, the struct will be merged into a single struct. E.g. `array.insert(0,['a']), array.insert(0, ['b']);` is the first represented as two structs (`[{id: {client, clock: 0}, content: 'a'}, {id: {client, clock: 1}, content: 'b'}`) and then merged into a single struct: `[{id: {client, clock: 0}, content: 'ab'}]`.

2. When a struct that contains content (e.g. ItemString) is deleted, the struct will be replaced with an ItemDeleted that does not contain content anymore.

[2] https://hal.inria.fr/hal-01248270/file/LS-consistency-ladis-2009.pdf
[3] https://www.researchgate.net/publication/310212186_Near_Real-Time_Peer-to-Peer_Shared_Editing_on_Extensible_Data_Types
[4] https://github.com/yjs/yjs#Yjs-CRDT-Algorithm

3. When a type is deleted, all child elements are transformed to GC structs. A GC struct only denotes the existence of a struct and that it is deleted. GC structs can always be merged with other GC structs if the id's are adjacent.

Ysj also has the ability to exchange only the differences when syncing two clients. It uses lamport timestamps to identify structs and to track in which order a client created them. Each struct has an struct.id = { client: number, clock: number} that uniquely identifies a struct.

2 About the Project

This project is a React web application that provides two real-time collaborative Quill editors. It supports synchronization in editing rich text that contains text, figures, and even videos. It also provides the saving function for users to save HTML output of the editor. Please notice that the saving function applies Blob API, which relies on the Browser compatibility. You can find the information for compatibility **here**. For this reason, browsers like Firefox for Android and Safari on iOS are not able to use this saving function, but the collaborative editing would perform properly on those browsers. The latest version of Chrome is encouraged to use for getting the best experience.

2.1 Project Structure

The project structure can be represented as:

```
root
├── public
│   ├── favicon.ico
│   ├── index.html
│   └── manifest.json
├── src
│   ├── App.css
│   ├── index.css
│   ├── App.js
│   ├── ConnectInfoArea.js
│   └── QuillEditor.js
├── Procfile
├── package.json
└── server.js
```

App.css and index.css define the layout of the application. QuillEditor.js defines the class of the real-time collaborative editor based on rich text editor Quill. More information about the Qill editor can be found at **here**. ConnectInfoArea.js defines the class of the connection information area that shows the current connected client numbers and their information. Building instruction details for the project can be found in package.json.

2.2 Project Setup

To Set up the project, run the following commands:

```
$ git clone https://github.com/camelboat/6770_ELEN_Cloud_Project
$ cd 6770_ELEN_Cloud_Project
$ yarn
$ yarn build
$ yarn serve
```

The client will then be running on localhost:3000. You can change the set the preferred server address by changing the address value in App.js. With default setting, the server is running on my AWS EC2 instance with the IP address of `http://3.136.85.49/`.

2.3 Project Demo and More information

You can find the project demo at **here**. Please visit the demo from different devices under different networks to observe the real-time collaboration.

Source code and detailed information of the project can be found at its **repository**.