CS651 - Grad Project Report

Collaborative Text Editor

Abhyudaya Alva
Gautam Bhat

Abstract

We have built a collaborative plain text editor, which allows multiple users to edit a document simultaneously on different machines on a network. The system that we have developed works based on the ideas and algorithm presented in the Jupiter[1] paper, which presents an optimistic concurrency control algorithm for collaborative editing.
The editor supports two main operations- INSERT and DELETE.
The following are the central ideas of our system.

1. All changes are first applied locally.
2. The changes are then transferred to the remote client with minimum delay.
3. Clients have the ability to detect conflict and resolve them in a reasonable manner.
4. The system could be made fault tolerant by using Replicated State Machines.

Design

The Collaborative Editor prototype has been built with the Java Swing framework, and we have used to socket-based communication between clients. Our eventual goal was to begin coding and testing out the Operational Transformation algorithm presented in [1] in a language we are familiar with, and then port it to a web-based service such as a web application hosted on a Node.js (or similar) server.
The users' edits become immediately visible, which are then forwarded to the remote location via sockets. A user can insert and delete characters from the text area. Each insert or delete is treated as a separate operation and propagated to the remote client. When the remote client receives the update, it has to be transformed so that it does not conflict with the local updates of the client. Updates are transformed using Operational Transformation, as discussed earlier.

To detect conflicts, we maintain a version number for updates for each user. The protocol labels each message with the state the sender was in just before the message was generated. The concurrency control algorithm uses these labels to detect conflicts, and applies an 'xform' function (explained below) to resolve them. The algorithm guarantees that, no matter how far the client and server diverge in state space, when they do reach the same state, they will have identical values.

The Jupiter paper[1] presents the idea for Operational Transformation as an xform function as shown below:

xform(c,s) = (c',s')

If user1 applies c and user2 applies s to their local document concurrently, then if user1 applies c' and user2 applies s' they both arrive at the same end result. Using the xform method we can transform all operations at the client end and apply so that the document remains consistent across multiple client.

The Jupiter paper provides the transformation method to only one of the xform methods. i.e When user1 and user2 both apply a delete operation and when those operations conflict. Our implementation provides the transform methods for other possible conflicting scenarios mentioned below:

1. User1 performs an insert and User2 performs delete.
2. User1 performs delete and User2 performs insert.
3. User1 performs insert and User2 performs insert.

Implementation

Our Java implementation provides collaborative editor for two users on two different systems. We have 3 main classes.

1. GUI
2. User
3. Message

We have provided a Java Swing based GUI and all the functionality related to GUI is handled in the GUI class. The GUI class creates a JFrame window with text component. A swing text component users a document to represent its content. We implemented a document listener to get notifications when the user makes any changes to the text area. We create appropriate messages (Message Class)and forward the updates to the other clients via the User class.

Message: The Message class represents a message that is to be passed between different clients. The fields myMsg and otherMsg represent the sequence numbers of local and remote client respectively. These numbers represent the state that the User was in before he applying an operation and as such these numbers are essential to detect conflicts and perform Operational Transformation.The four xform methods are also provided as static methods in the Message Class. The XForm methods are used by our User class to transform conflicting operations.

User: The User class handles the communication between the User and the Client and also takes appropriate action when a communication is received. The User class acts as the intermediary between the GUI handler and the remote client. In order do this the User class requires 4 essential fields. Each User opens two socket connection. One Server socket on which it listens to Messages sent from the other User. One user socket through which it sends the updates to the remote client.

IP address of the remote client
Port number of the remote client
Port number to start the server socket.

The Socket read and write are performed on separate threads.

The User class performs two primary actions:

1. On receiving an update from GUI, it adds the Message to the queue and forwards it to the other client.
2. On receiving a message from the remote client, it transforms the message if the remote client has not applied the operations that the current user implemented. Operations that have been performed at both ends are deleted from the Queue since they are guaranteed to be consistent.

Challenges Faced

Like stated before, our most optimistic goal was to implement a web application-based collaborative text editor, that could serve multiple users editing plain text document (where multiple means, two or more). However, while the idea of operational transformation is intuitive and seems quite simple to grasp once understood, it's not that simple and straightforward when it comes to implementation, and maintaining synchronized documents across multiple users. While the paper [1] claims that the model described for two users can easily be extended to more than two users, we found it difficult and not-so-easy to implement. Moreover, while porting and translating our operational transformation to a server-based javascript web application model, we faced several bugs and technical difficulties along the way due to unfamiliarity with the tech stack. As of writing, we are still working on and trying to complete the implementation of

the web application in time for our presentation, without any bugs, while delivering our minimum viable requirements.

Conclusion

From a distributed systems point of view, we have implemented a collaborative text editor, that allows two (or hopefully more) users to collaboratively edit a common plain text document from their own machines, over a network, while ensuring consistency among the versions as long as the two machines can communicate over the said network. The idea allows for clients that are disconnected to operate independently and synchronize. The system can become fault tolerant if the server-based implementation uses an RSM, such as we saw in Raft and Paxos.

References

[1] David A. Nichols, Pavel Curtis, Michael Dixon, John Lamping, *High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System*, UIST 95' Pages 111-120, 1995