"ParallelPad"

The Straightforward Collaborative Text Editor

Dipen Gupta and Bharadwaj Mudumba

PennState
Harrisburg

A project proposal for COMP 512

Department of Math and Computer Science
Penn State Harrisburg

Table of Contents

1 Introduction

Collaborative editing is a really cool and genuinely useful feature to gain traction in the past decade. It is the idea of multiple people being able to write on and make simultaneous edits on a single, shared file. Popular examples of this include online applications like Overleaf [1] and Google Docs. [2] Usually, this includes creating (or uploading in some cases) a file online, and "sharing" the same with another person via a link. The other person is then able to make simultaneous edits on the same file. Multiple users are generally distinguished by different coloured cursors.

Presently, there is no way to prevent multiple people from editing on the same line at the same time, which can get quite annoying at times. Our idea here is to modify this such that if two users, say user1 and user2 are working on a file, and if user1 is writing a line, user2 should not be able to make edits on the same line. This, we think will result in a better user-experience.

2 Problem Statement

All collaborative text editors aim for conflict-resolution, and make use of algorithms like Operational Transformation (OT) [3][6], Conflict-free Replicated Data Type (CRDT) [4][6], or some version of them (e.g., Transparent Adaptation [7]) which help with the goal of eventual consistency in the document.

> "However, specifications of their desired behavior have so far been informal and imprecise, and several of the protocols have been shown not to satisfy even the basic expectation of eventual consistency" [5]

We, however propose a different approach wherein we do not let multiple users make edits on the same line. Not only does it make the task of maintaining consistency more straightforward, but also it arguably provides a better user-experience as there is no confusion when users might accidentally make edits on the line where someone else is writing.

Making such an editor will require use of core concepts and skills of Distributed Systems. We plan on using a client-server model for this idea, and want to design the system to be scalable, concurrent, transparent, heterogeneous while providing a good quality of service and being fault tolerant.

3 Design

3.1 Overview

Our system will be based on the Client-Server architecture, and will make use of SFTP and TCP protocols. The clients are the individual users who want to perform write and make edits. Each client will have a local copy of the document. The server will be concurrent and stateful. It will have the roles of maintaining the master copy, and managing the requests from clients.

We plan on using locks to develop our solution of two users not editing on the same line. Whenever a client wants to perform an edit, they make a request for 'write' lock to the server, and the same will be released when it finishes editing. The updated copy of the document is then sent to the server, which becomes the master copy. This master copy is sent to all the clients to update their document. Server can issue a 'read' lock to any number of clients at a time (i.e., clients should be able to read the document at any given time.)

When the server gets request for a 'write' lock:

- If 'write' lock is not issued to any other client (i.e, no other client is editing on that line) the same is issued to the requesting client.
- If 'write' lock has already been issued to some other client, the server asks the requesting client to wait till the 'write' lock is released, giving a message similar to "WRITE IN PROGRESS BY OTHER CLIENT".

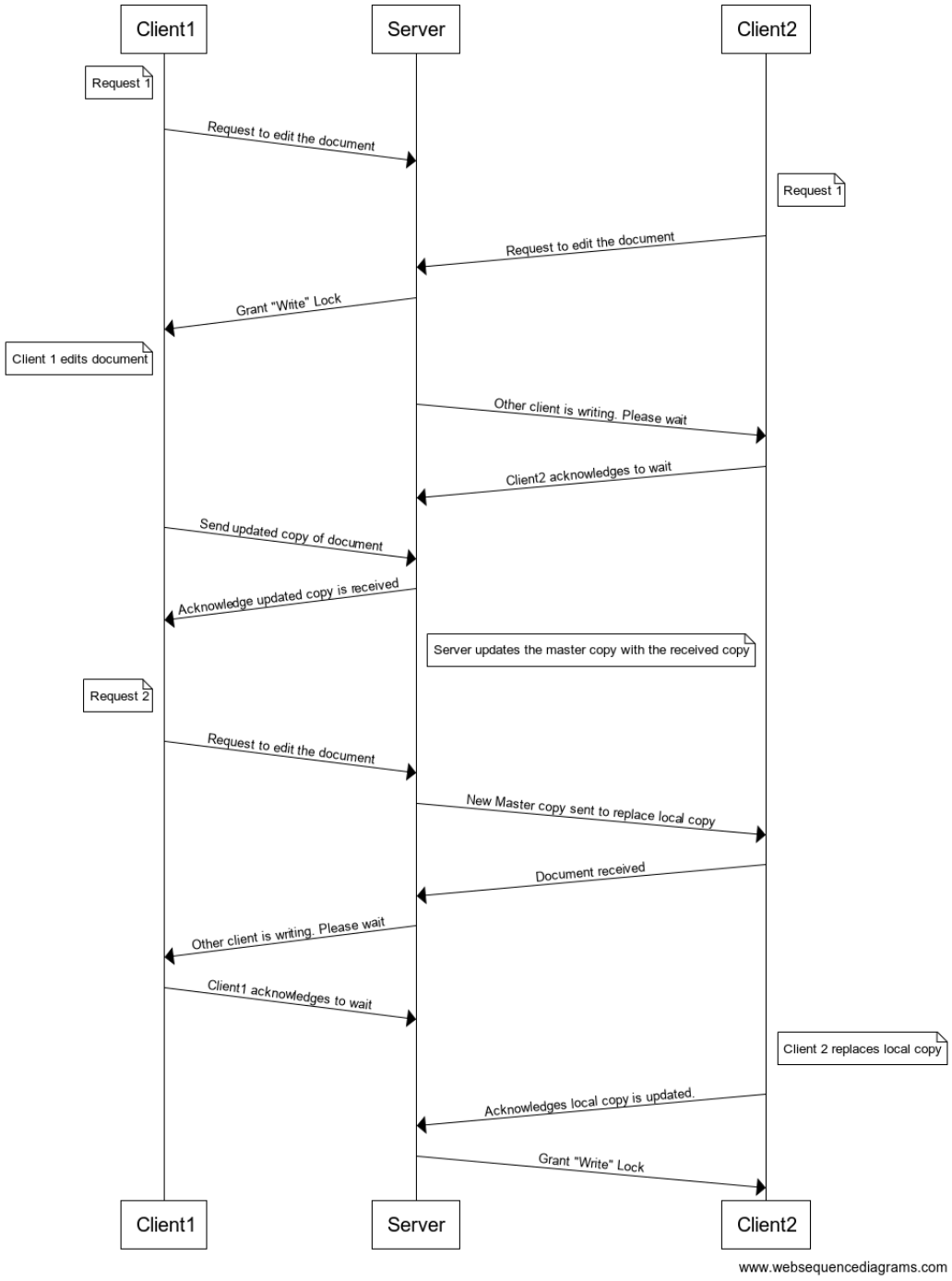

Figure 1: Sequence diagram for requesting and granting "write" lock

When the client is done editing:

1. they will send the "Updated Copy" to the server, which will be acknowledged by server.
2. This "Updated Copy" will become the "Master Copy" on the server, and will be sent to all other clients, who will acknowledge this by returning a "received".
3. The clients will replace their local copy with the updated copy, and send an "updated" message.
4. On receiving these "Updated" messages from all clients, server will issue the 'write' lock to another client in the order of request received.

3.2 Failure Handling

- To handle to scenario where there is no activity at client, server will implement a timeout mechanism . If there is no 'heartbeat' for 10 minutes, we consider the client to be down and revoke the 'write' lock. 'Heartbeat' here refers to a message that is sent from server to the client, which the client has to acknowledge and reply to.
- If there is a server crash, the information before the crash should be retained. A stateful server will be implemented for the same.
- In the event of a crash on the client's end, autosave mechanism will be implemented, and their changes will not be lost.

3.3 Starvation Avoidance

- We ensure that no client will suffer starvation. If any client is waiting for "write" lock for more than 30 minutes. Server sends a message that "Write permission cannot be obtained at the moment, retry after a while".

4 Project Timeline and Scenarios

4.1 Project Milestones

- (2nd week of February) Evaluate technical requirements and system architecture.
- (2nd week of March) Have a basic collaborative text editor ready. (i.e., the skeleton)
- (last week of March) Implement the solution.
- (2nd week of April) Test thoroughly and bug fixes.
- Documentation and work on the project report will be done along the way.

4.2 Project Scenarios

- Minimal: We get a collaborative text editor up and running, and multiple people are able to edit on the same file.
- Expected: Minimal case, along with our idea of not letting two users edit on the same line.
- Best-case: We are able to host this project online with a unique link for every document, and users should be able to access it from any device.

References

[1] https://www.overleaf.com

[2] https://www.google.com/docs/about/

[3] Junlan Liu, Guifa Teng, Yan Shao, Wei Yao and Sufen Dong, "Concurrency control strategy in real-time collaborative editing system," 2010 2nd International Conference on Education Technology and Computer, 2010, pp. V3-222-V3-225, doi: 10.1109/ICETC.2010.5529560.

[4] Beresford. 2019. Interleaving anomalies in collaborative text editors. In Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '19). Association for Computing Machinery, New York, NY, USA, Article 6, 1–7. DOI:https://doi.org/10.1145/3301419.3323972

[5] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. 2016. Specification and Complexity of Collaborative Text Editing. In Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16). Association for Computing Machinery, New York, NY, USA, 259–268. DOI:https://doi.org/10.1145/2933057.2933090

[6] Johannes Wilm and Daniel Frebel. 2014. Real-world challenges to collaborative text creation. In Proceedings of the 2nd International Workshop on (Document) Changes: modeling, detection, storage and visualization (DChanges '14). Association for Computing Machinery, New York, NY, USA, Article 8, 1–4. DOI:https://doi.org/10.1145/2723147.2723154

[7] B. Cho, A. Ng and C. Sun, "CoVim: Incorporating real-time collaboration capabilities into comprehensive text editors," 2017 IEEE 21st International Conference on Computer Supported Cooperative Work in Design (CSCWD), 2017, pp. 192-197, doi: 10.1109/CSCWD.2017.8066693.