O-PAD

A Collaborative Text Editor

Dushyant Panchal (2018033)
Osheen Sachdev (2018059)
Paras Mehan (2018062)
Adwit Singh Kochar (2018276)

Problem Statement

Distributed text editors are commonly used to support collaborative editing for software or document development. However, many existing solutions work with documents on the server; they do not support direct collaboration between two clients working with locally stored documents.

We propose a distributed text editor for real-time collaboration that stores the files locally.

This idea has been motivated from the following technologies:

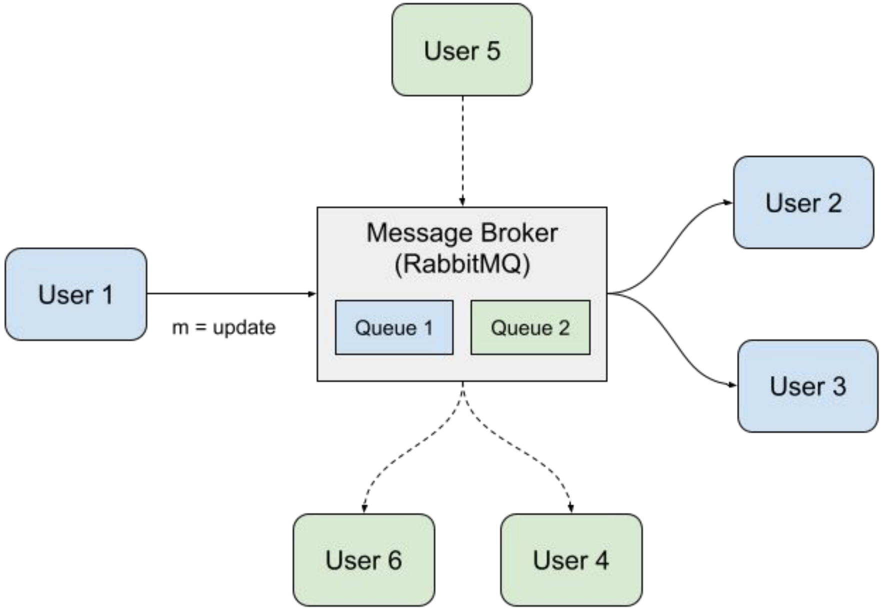
- Torrent
- Git
- VSCode

Architecture

Architecture

O-Pad Client: O-Pad client is the application running on the user side that manages the text editor and sends and receives updates regarding changed files.

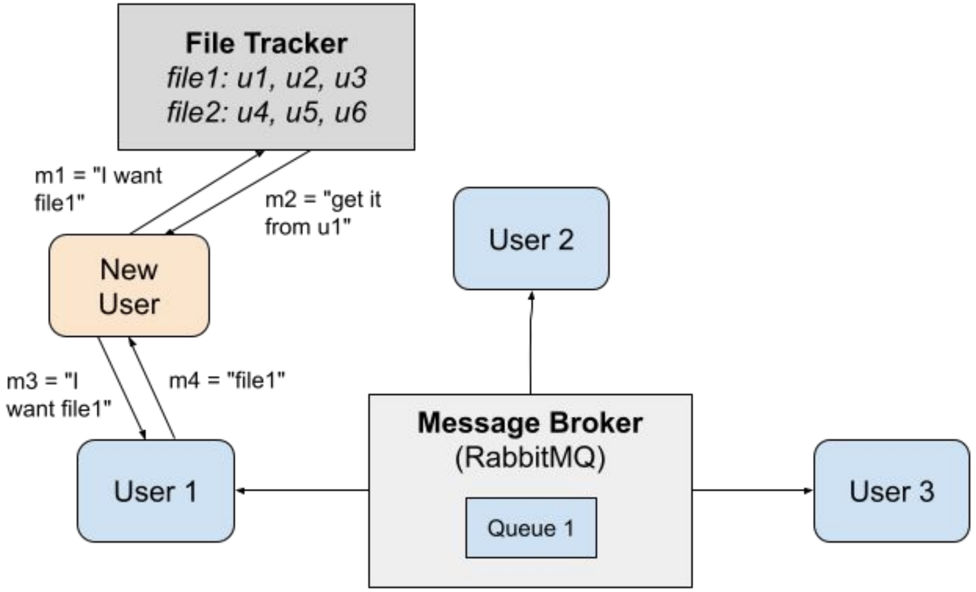
Message Broker: Maintains exchanges for each file being edited live and a queue for each user.



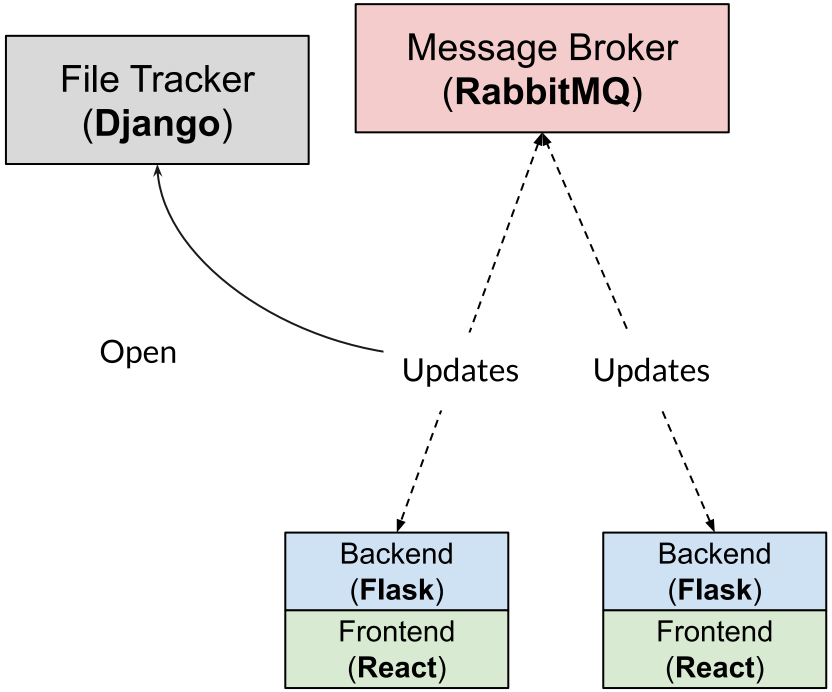
Architecture

File Tracker: Maintains a map of files and the list of online O-Pad clients working on that file, i.e. the collaborators. Facilitates joining for the first time.

RabbitMQ Server: manages message exchanges, for client-client communication.



Tech Stack

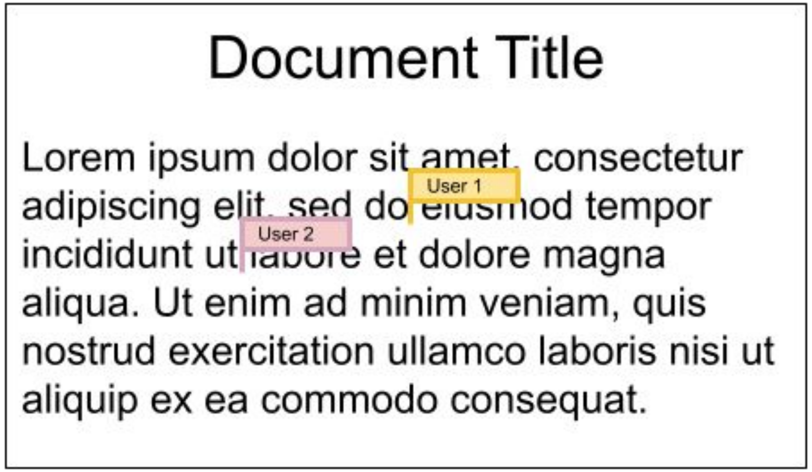

File Tracker
(**Django**)
Message Broker
(**RabbitMQ**)
Open
Updates
Updates
Backend
(**Flask**)
Frontend
(**React**)
Backend
(**Flask**)
Frontend
(**React**)

Challenge 1: File Consistency

Problem: Multiple writes

- **Handling concurrent writes:** multiple users will modify the file from different systems.
    - We need to synchronize the clocks of these systems so that the updates are sequentially consistent.
    - Since we are only sending deltas, we need to have a concrete algorithm for resolving conflicts for concurrent updates.
- **Asynchronous receive:** Changes to the file will be resolved without any intervention of the user using asynchronous communication (unlike git).

Merging Updates


Document Title

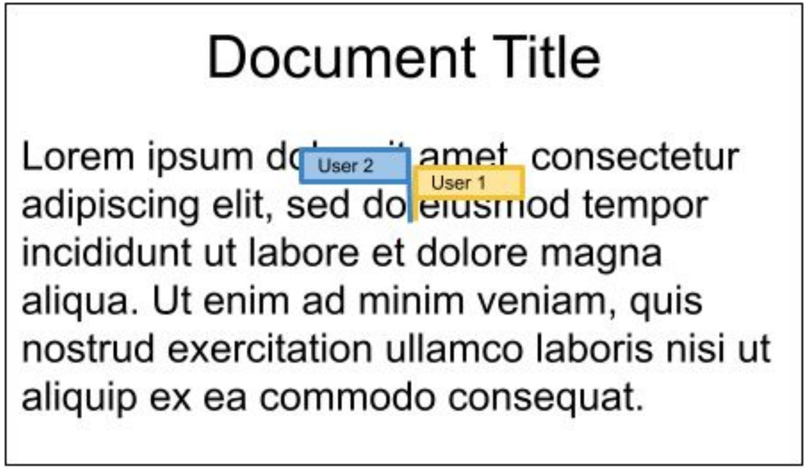
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

User 1 types "a" and User 2 types "b" at their respective cursor positions simultaneously.

Update Messages Generated:
insert a at position 31
insert b at position 46

User 3 receives these two updated messages.

Sequential Consistency



Document Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

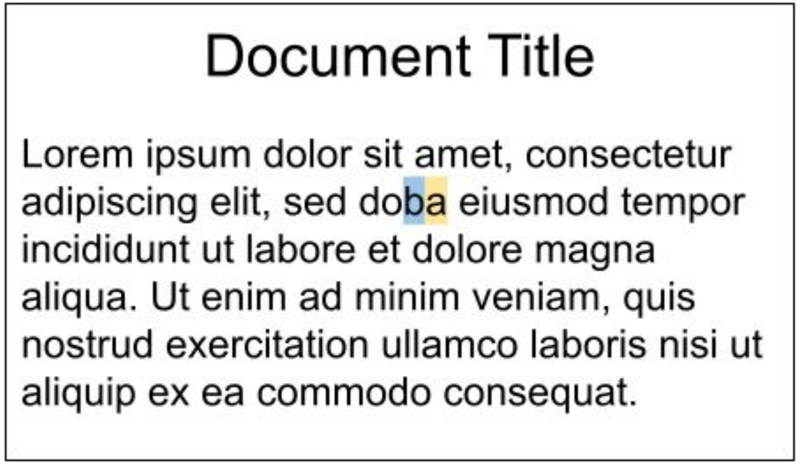
User 1 types "a" and User 2 types "b" at their respective cursor positions simultaneously.

Update Messages Generated:
insert a at position 31
insert b at position 31

User 3 and User 4 receive these two updated messages.

Sequential Consistency

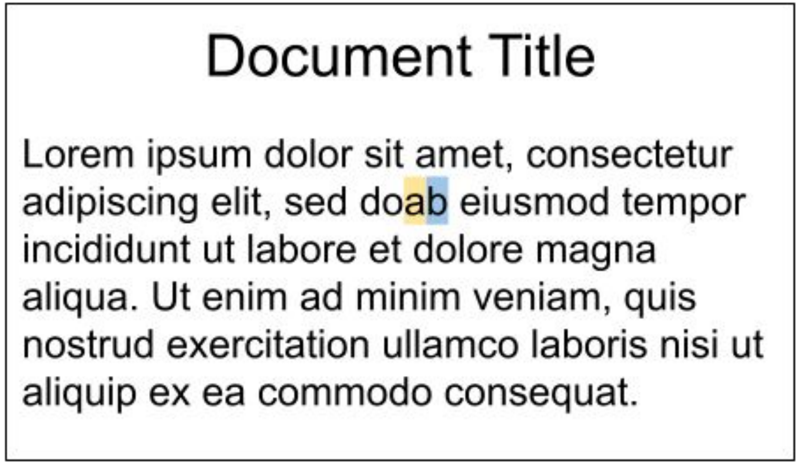
User 3



Document Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed doba eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

User 4



Document Title

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed doab eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Solution: CRDT

Conflict-Free Replicated Data Types makes all operations **commutative** in the following ways:

- Each character has a unique **positional identifier**
  - Generated using floating point numbers.
- Each user uses a Lamport clock.
  - To resolve conflicts
- Each character has the user ID along it.
  - Helps distinguish same positional identifiers from different users.

Challenge 2: Cursor Consistency

Frontend-backend synchronization

Since frontend and backend are running on different processes, we also need to synchronize them.

Original File:

"a|"

insert b at position 1

Since backend hasn't responded on success of insert and updated file "ab|", the cursor stays on "a|".

Now the user types c.

insert c at position 1 | insert c at position 2

"acb" | "abc"

Shifting Cursor on Receiving Changes

CRDT doesn't apply to the cursor position, hence it is not updated when another user types text before the cursor. For example, with

Original text:

"aaaa|"

An insert is made at position 1 by another user.

"baaaa|"   "baaa|a"

Solution

We implemented our own cursor

- Ensure only one insert or delete happens at a time
- Cursor updated by backend to avoid frontend-backend synchronization
- Move the cursor on insert or delete of character by another user by identifying the position of insert.

Challenge 3: Recovering from Crashes

Challenge: Recovery

When client crashes and restarts, previously open files are closed according to the backend but open according to the file tracker.

How to notify the file tracker that the client no longer has the file?

Solution

- **Passive**: polling from the filetracker to every client every 10 seconds. If the client responds with not having the file open, the file is closed. If the client doesn't respond within 2 seconds, the client is considered dead.
- **Active**: when the client receives a request from another client for a file it does not have, it sends a close request to the filetracker
- **Balancing**: Rotation of File tracker entries for a particular file so that the same failing entry is not returned on every request

Thanks!

Any questions?