

“ParallelPad”

The Straightforward Collaborative Text Editor

Dipen Gupta and Bharadwaj Mudumba



PennState
Harrisburg

Project report for COMP 512

Department of Math and Computer Science
Penn State Harrisburg

Table of Contents

1	Introduction	3
1.1	Motivation/Problem Statement	3
1.2	Literature Survey	3
2	Distributed System Concepts in ParallelPad	3
2.1	Architecture	3
2.2	Deadlock Prevention	3
2.2.1	Hold and wait	3
2.2.2	Circular wait	4
2.2.3	Mutual Exclusion	4
2.3	Message Semantics	4
2.4	General Rules	5
3	Functional Implementations	5
3.1	Main Logical Flow	5
3.1.1	Part 1: Requesting and granting a lock	5
3.1.2	Part 2: Releasing the lock	5
3.2	Handling Cases - Popups!	5
3.3	Crashes	6
4	Technical Implementations	9
4.1	Special Data structure “Chain”	9
4.2	Backend - Java	10
4.3	Front End/UI - Java Swing	11
4.4	Database - Oracle SQL Plus	11
5	Challenges Faced	11
5.1	Architectural Challenges	11
5.2	Network Challenges	11
5.3	UI Challenges	11
6	Conclusion	12
6.1	Evaluation	12
6.2	Future Work	12
	References	13

List of Figures

1	Main Logical Flow - Part 1	6
2	Main Logical Flow - Part 2	7
3	Server figuring out that client crashed.	9
4	Client getting notified that the server crashed.	9
5	Sample Chain Data structure	10

List of Tables

1	Message Semantics.	4
2	Popups!	8
3	Chain Operations	10

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.

1.1 Motivation/Problem Statement

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.

1.2 Literature Survey

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

“However, specifications of their desired behaviour 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]

However, we 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.

2 Distributed System Concepts in ParallelPad

2.1 Architecture

Our system is based on the **client-server architecture**, over **TCP protocol** using **Java Sockets**. It makes use of Oracle’s SQL Plus database. Clients are the individual users who want to edit the file. Each client has their local copy of the document which they can edit. Multiple clients can concurrently work on the document, with the changes coming up even as they are editing. The server is **stateful**. It retains the last saved copy of the document. In addition to this, it has the roles of maintaining the master copy, keeping track and coordinating between clients and managing all the requests that come in. The main idea of not letting multiple users edit on the same line is implemented by making use of **locks**. A custom data structure “Chain” is employed to help with this. (More details are present in section 4.1)

2.2 Deadlock Prevention

We make sure that the system doesn’t enter deadlock as there is no hold and wait, circular wait and resources are accessed mutually exclusively.

2.2.1 Hold and wait

A client can request a lock only after releasing the existing lock. For example, if user 1 has to request for a lock held by another user 2, the lock held by user 1 will be released before requesting for user 2’s lock. This ensures that a client does not hold a lock and also waits on another lock.

2.2.2 Circular wait

As hold and waits are not possible in our system, there is no chance of a circular wait. For example, if user 1 wants lock held by user 2 and vice-versa. Whenever any user makes the request, their current lock will be released and the other user gets it. This ensures there are no circular waits in our system.

2.2.3 Mutual Exclusion

As it is a distributed environment, multiple users have to edit at the same location in the document, granting access to multiple users will result in the issue of overwriting resulting in inconsistency in the system. We have effectively handled this situation by implementing **centralised mutual exclusion** such that the server will make sure that one user can edit an object at a time.

- **Safety** We ensured safety property such that the server will provide permission to edit an object in the document if no other user is currently editing that object. If any user is editing that object, the server alerts the user to wait via a popup.
- **Fairness** The server maintains an ordered list of requests for each tokenNode of the document in the chain and only grants the permission to the user (i.e., clientNode) who is at the top of the list. The first user to request to edit will first get permission to edit. This ensures that fairness is guaranteed. (PS: We have ignored the communication channel delays as that hasn't caused any issue without implementation.)
- **Liveness** Unless the user has left the network, the user will be granted the permission to edit the object. This guarantees liveness. However, if the user has requested an object and while waiting for that object has requested some other object, the previous request is invalidated and a new request is made.

2.3 Message Semantics

As we know, any communication in a distributed system happens by passing messages. In our project, we have defined a particular protocol via which any message going through the network will follow. Else, we do not process it.

Two types of messages can be sent:

- **M1 (*Code message*)**: this describes the specific action that has happened and the actions that need to be done. M1 messages always begin with parameters for $\langle Code \rangle$ and $\langle ClientID \rangle$. These and other parameters are delimited by "&" symbol. The complete list is provided below.

Code	Meaning	Initiated by
0	Heartbeat	Server
1	Initial connection request by client	Client
2	Client requests lock from the server by clicking in the UI	Client
3*	Server grants lock to client	Server
4	Server tells the client to wait for lock	Server
5	Client is done editing and clicks the save button	Client
6	Server sends the client the latest copy of the document to update locally	Server
7	Acknowledgement of Code 6	Client
8**	Server releases client's lock	Server
9	Client sends a message to close connection	Client

Table 1: Message Semantics.

*Code 3, has special sub codes 0, 1 and 2.

- Meaning of 0: A client asked for the lock and got it immediately.
- Meaning of 1: Client had to wait for the lock and is getting it now.
- Meaning of 2: The lock for which the client was waiting has been deleted.

****Code 8**, has special sub codes 0 and 1.

- Meaning of 0: Regular case to release the lock.
- Meaning of 1: This is to handle the case where client has deleted all text for their lock.

- **M2 (*Document Message*)**: this essentially sends the entire document over the network, line-by-line. This can only be sent after an appropriate M1 and never by itself.

Note: M2 is sent only after Codes 1, 5, 6 and 8 (sub code 1).

2.4 General Rules

Users must be aware of the following “rules” of ParallelPad:

- *Users must click on the line they want to edit.* As mouse-click requests for lock, without clicking, the user cannot enter any text.
- *Users must click the save button after they are done making their edits.* The whole flow for updating the database and the copies of other users are triggered by this.
- User must not use arrow keys or the mouse to go to another line/area that they do not have the lock for. Doing so will display a popup message.
- Any time that some user saves their document, the change is applied immediately, even when the user is working on some other part of the document.

3 Functional Implementations

This is an easy to use application, even for a novice user. In essence, the user can simply click anywhere in the document and write. The system handles everything behind the scenes and updates the user through popups when the need arises.

3.1 Main Logical Flow

There are two parts to the main flow that happens in the application. This makes up the main chunk of the application and anything that branches out of this flow is handled separately.

3.1.1 Part 1: Requesting and granting a lock

This is initiated by the user clicking somewhere in the UI. The request for that particular lock is then sent to the server. The Server checks the chain, updates it and then sends a reply to the client.

In the sequence flow diagram (fig. 1), we have two Clients, Client 1 and Client 2. Client 1 requests for a lock and gets it. Client 2 then requests for the same lock and is not granted the lock. The chain is updated in both cases.

3.1.2 Part 2: Releasing the lock

This is initiated when the user clicks the save button. The client’s copy is sent to the server and the server updates the database, sends all the other clients the new copy and after getting acknowledgements from all other clients, it updates the chain and then releases the lock.

Each client, upon receiving the document will update its local copy with the received copy. This ensures **consistency** is maintained across the system.

Note: the sequence flow diagram (in fig. 2) is not a continuation of Figure 1, but if it had been the server would send a Code 3 to Client 2 at the end, if it’d been waiting for Client 1 to finish editing.

3.2 Handling Cases - Popups!

Table 2 describes how different scenarios that branch out from the main flow are handled. All possible popups are described there, along with their cause and the next steps that the user can do after receiving it.

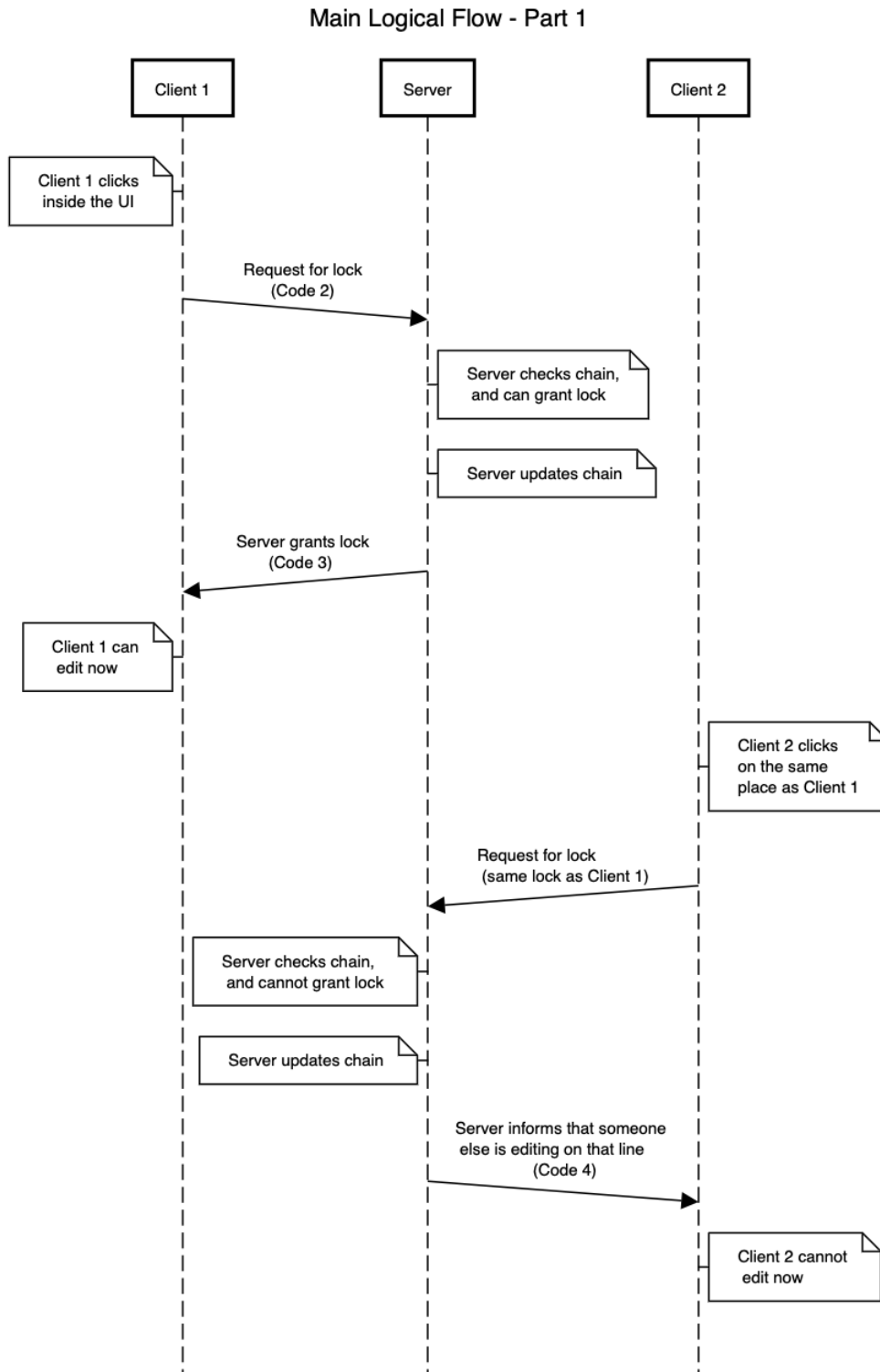


Figure 1: Main Logical Flow - Part 1

3.3 Crashes

- **Client Crash:** The Server detects a client crash if they do not respond to the heartbeat. If a client was waiting for the lock held by the crashed client, they will get Code 3.
- **Server Crash:** The Client gets a popup notifying them that the server crashed. The last saved copy is retained with the server.

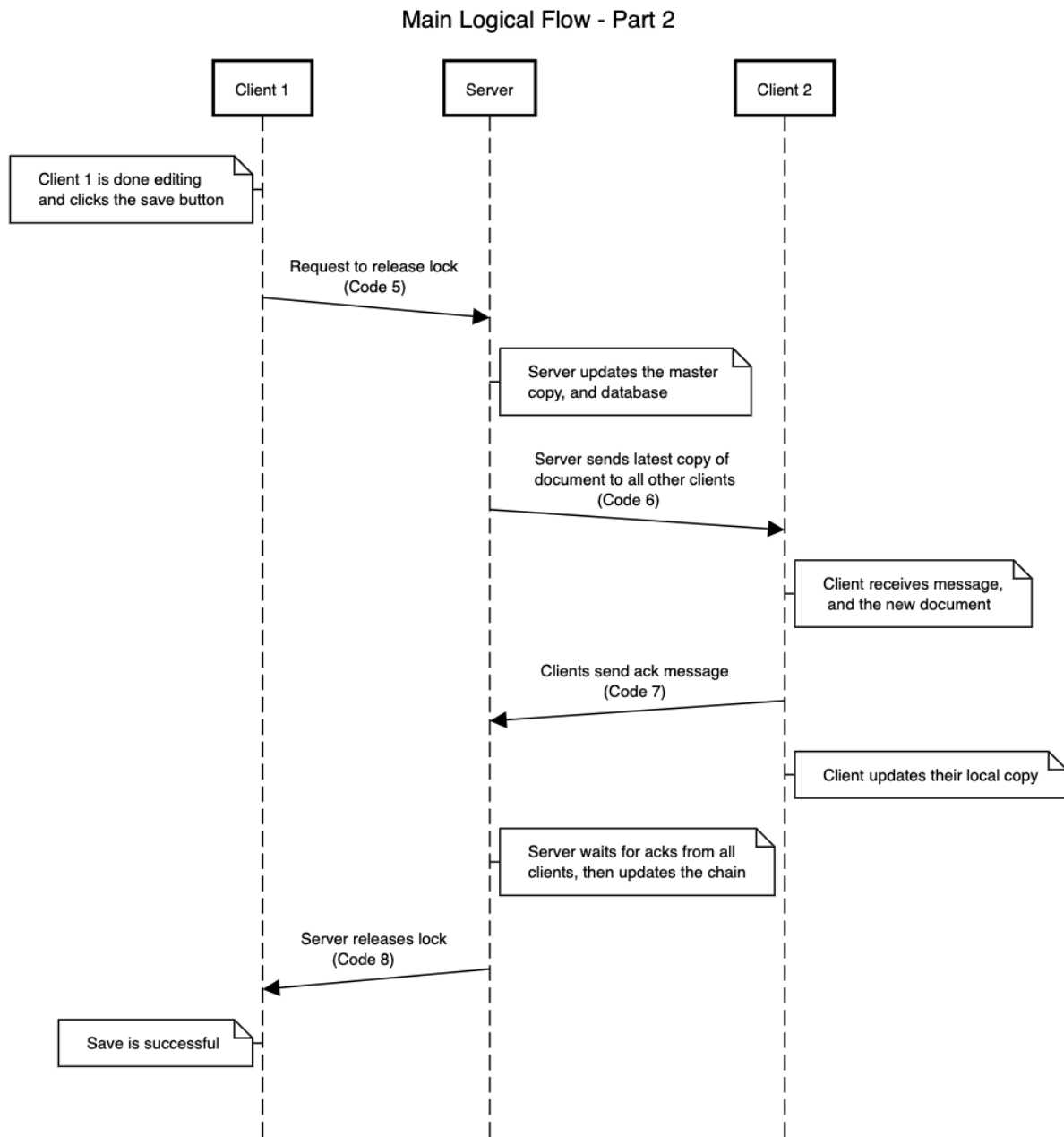
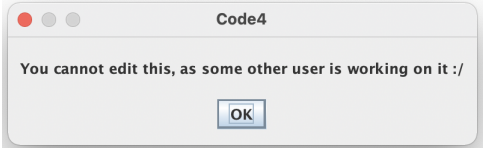


Figure 2: Main Logical Flow - Part 2

Popup	Cause	Next Steps
	Some other user is editing on that line/area.	The User can either wait, or click somewhere else in the UI.

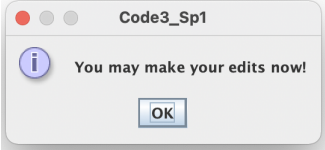
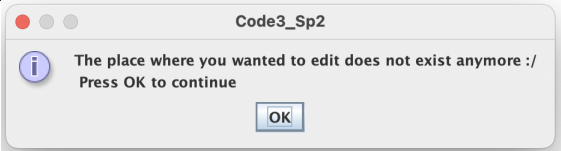
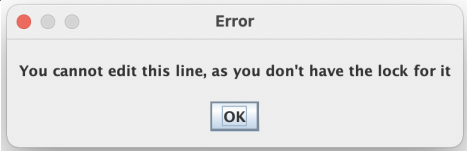
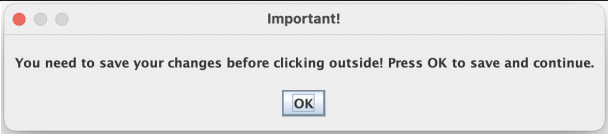
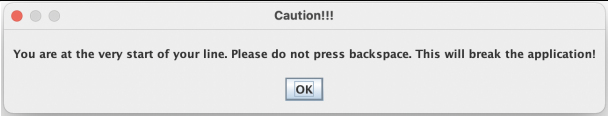
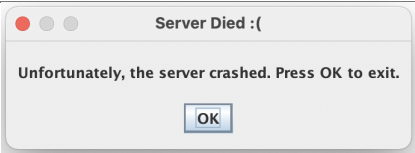
	<p>This comes after you decide to wait on receiving a Code 4 and the lock has been released.</p>	<p>The user can make the edits, or click somewhere else in the UI.</p>
	<p>This comes after you decide to wait on receiving a Code 4, but the current editing client has deleted everything.</p>	<p>User can click somewhere else in the UI.</p>
	<p>User uses arrow keys or mouse to go to a line which he does not have a lock.</p>	<p>User can click OK and edit on the line for which they have the lock for.</p>
	<p>User made some changes, but did not click save before clicking on another line.</p>	<p>User can click OK and continue.</p>
	<p>User comes to the very first character by clicking backspace.</p>	<p>User can click OK and continue editing, being careful not to delete before the first character.</p>
	<p>There is some issue at the server and it crashed.</p>	<p>Users have to click OK and exit the application.</p>

Table 2: Popups!

```

-----
Client 41864 seems to be down! Sending message again to confirm
Client 41864 seems to be down! Sending message again to confirm
~~~~~IMP Heartbeat notification start~~~~~
Removing client 41864 from activeClients
Here is the list of connected clients[49448]
****CHAIN TRAVERSE****
Chain's Sentinel Node: T1
Chain's TokenNodes: T1 T301 T1501

Client nodes for T1:

Client nodes for T301:

Client nodes for T1501:

****CHAIN TRAVERSE END****
~~~~~IMP Heartbeat notification end~~~~~
message from client: 0&49448
-----
message from client: 0&49448

```

Figure 3: Server figuring out that client crashed.

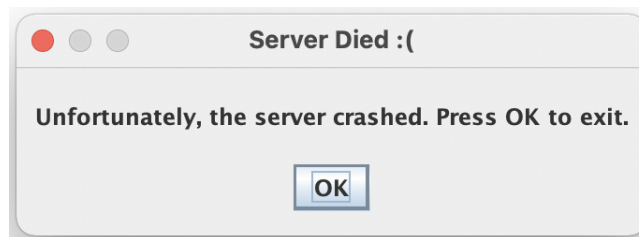


Figure 4: Client getting notified that the server crashed.

4 Technical Implementations

4.1 Special Data structure “Chain”

We have come up with a special “chain” data structure to handle locking and maintaining document consistency. The Chain comprises of:

- “*TokenNodes*”: “Tokens” here refer to a “lock” that is associated with every lockable object present in the system (i.e., a line in our UI). These are always maintained in the order that they appear in the document. Tokens can be inserted or deleted based on the edits made by clients.
- “*ClientNodes*”: These contain the ClientIDs who have the lock and other clients who are waiting to get the lock. The first ClientNode dangling from the TokenNode has been “granted” the lock and only that client is allowed to make edits on that line.

A “SentinelNode” here is just a special pointer that always points to the first TokenNode (i.e., the first line in the UI). Each TokenNode, can have a list of ClientNodes dangling from it. The main idea here is to keep a track of the tokens/locks present in the system, the order in which these “tokens” appear in the UI and the order of the

clients waiting on these “tokens”.

Figure 5 below describes a sample chain, which is keeping track of 6 clients working on a document with 4 lines. From this figure, we can say that no one is editing on line 2 (with TokenNumber T2). Client 1 is editing the first line (which has the TokenNumber T1) and Client 2 is waiting for a lock held by Client 1. Similar conclusions can be drawn for lines 3 and 4.

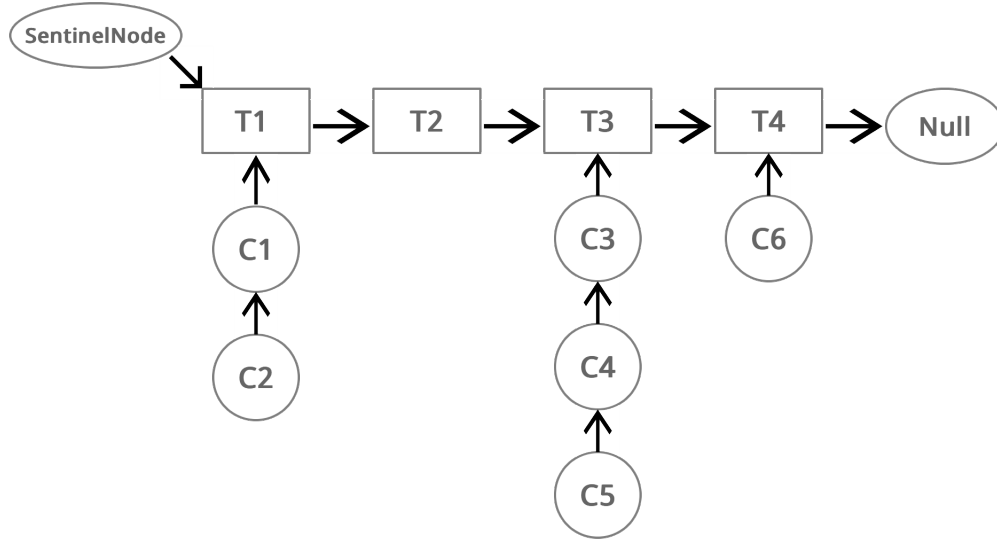


Figure 5: Sample Chain Data structure

The Chain can change in the ways outlined in Table 3.

Case	Action performed by client	Change in the chain
1	Client clicks on a line to edit on it.	ClientNode is added to the respective TokenNode.
2	Client clicks save after editing and the edits does not add new lines.	No new TokenNodes are added.
3	Client clicks save after editing and their edits results in creation of multiple lines.	Depending on the number of lines created, the same amount of new TokenNodes are added.
4	Client clicks save after editing. If the edits result in deletion of the line.	Corresponding TokenNode is deleted.
5	Client is waiting for a lock and clicks on the same line.	There is no affect in the Chain.
6	Client is waiting on a lock and clicks on another line, which has no clients dangling from it.	ClientNode is deleted from the previous TokenNode and added to the new TokenNode.
7	Client is waiting for a lock and clicks on another line, which has multiple clients dangling from it.	ClientNode is deleted from the previous TokenNode and added to the end of the new TokenNode.

Table 3: Chain Operations

4.2 Backend - Java

Our project is entirely written in Java which enabled us to use its rich set of libraries and data structures for all our needs. The general idea for the backend is to have a server, which forks itself to handle any client that connects. (i.e., create a new Java “Thread” to handle the client) [8]

Java “Sockets” are used to create sockets, establish connections, pass data and eventually close the socket. We have made use of Java’s “ScheduledExecutorService” to implement the heartbeat feature. [9][10]

4.3 Front End/UI - Java Swing

We have used Java Swing to create the text editor. [11] User can read the document at any time. However, requires a lock to edit the document as the text editor is non-editable by default.

We have created a Mouse Listener, if a user has to edit the document he has to request the server for the permission to edit the document this can be done by clicking at a location in the document where he has to do the edits. This mouse event will trigger the lock request to the server, which is part 1 of the main logical flow.

Users with a lock can release it by clicking the “Save” button, which will trigger part 2 of the main logical flow.

4.4 Database - Oracle SQL Plus

The server is linked to the Oracle SQL Plus database using a JDBC connector. We are maintaining two tables. The first table contains a list of tokens, along with the timestamp. The idea with the timestamp is to fetch the latest token list when the server starts. The second table contains the tokens and the value associated with that token. This is exactly like storing the lines of the document along with its token id. Prepared statements are used for querying the database from the server.

The tokens and the values are logged in the database whenever the user releases the lock. In the event of a server crash or failure, the most recent logs are used to recreate the document.

5 Challenges Faced

5.1 Architectural Challenges

- The server creates a new thread (ServerChild) to handle any client that comes in. We needed a way for the ServerChild to be able to carry out tasks in the Main Logical Flow. This would require it to access functions and variables that are declared and maintained by the server. The solution to this was to make those required functions and variables “Static”.
- We need a token id for every “line” that would be present in the document. Since the token id acts as a unique identifier, we needed a way such that no two clients use identical token ids. How we got around it was to give the clients a range of token ids to use with Code 1. So, each client, upon connecting to the server gets 200 unique tokens to use.

5.2 Network Challenges

- Client crashes: We implemented a “heartbeat” functionality, which sends out a Code 0 every 10 seconds, which the client is expected to respond with in 500ms. If the client does not respond to the first heartbeat, they are sent two more heartbeats, with intervals of 1.5 seconds. Even after that if the server does not get any response, that particular client is removed from the server’s list of clients.
- FTP: We had initially planned on implementing FTP to send the document across the network, but we were facing Java errors trying to implement it. The contents of the transferred file were also getting garbled. We then resorted to sending the file line-by-line via the already established Java Socket.

5.3 UI Challenges

- We update users via popups and initially while implementing the heartbeat feature, these popups use to block the code flow and thus the client was unable to send the reply to the heartbeat. The solution to this was to change the modality of the popup box and run that in a new thread.
- We faced multiple issues while trying to get text and cursor position and trying to pass it to the server. The major issues consisted of being unable to identify the end of the line, mouse click on the first character was getting a lock of the previous line. All these were solved by debugging the entire code and making logical changes.

6 Conclusion

The project was carried out successfully and a working application was made. Goals that were set in the beginning were met.

6.1 Evaluation

We were able to attain the goal of making a collaborative text editor, where two users are not able to edit on the same line. Core concepts of distributed systems like deadlock avoidance, mutual exclusion and client-server model were implemented in the project.

6.2 Future Work

Although ParallelPad handles a lot of scenarios, corner cases and does the job of collaborative text editing, the following can be implemented to improve the experience:

- Presently, a user can get a lock and doesn't perform any action. A mechanism to revoke locks after a long period of inactivity can benefit another user wanting to edit on the same line.
- Encrypting the messages sent over the network to make the application more secure.
- UI can be enhanced with rich text features. (e.g., bold, italics, underlines, font sizes, different fonts etc)
- Implementing a backup server, so even if the main server crashes, the clients can still continue with their work.
- Making a trigger to ensure that data saved in the two tables are consistent.

References

- [1] <https://www.overleaf.com>, last accessed on 9th April, 2022
- [2] <https://www.google.com/docs/about>, last accessed on 9th April, 2022
- [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.
- [8] <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>, last accessed on 9th April, 2022
- [9] <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>, last accessed on 9th April, 2022
- [10] <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html>, last accessed on 9th April, 2022
- [11] <https://docs.oracle.com/javase/tutorial/uiswing/>, last accessed on 9th April, 2022