



Winter 2018  
Distributed System COEN 317

# Bronco-Editor

## A real-time collaborative editor



Project By

Vaishali Ramesh Rao Dhulshette  
[vdhulshette@scu.edu](mailto:vdhulshette@scu.edu)

## Audience

This report covers the details of the real time collaborative editor which is implemented using the CRDT sequence algorithm. The primary audience for this paper are the programmers, students and anyone who need to understand the underlying concepts of distributed systems and need to try a practical implementation for the same. This report covers the challenges of the original problem of implementing the collaborative editor.

## Contents

Audience .....	2
Introduction .....	4
1. Goals .....	4
2. Distributed system challenges .....	4
2.1. Communication Latency: .....	4
2.2. Concurrent Edits .....	5
2.3. Challenge of Idempotent: .....	5
2.4. Synchronization between all the clients : .....	5
3. How the challenges are addressed .....	6
3.1. Conflict-free replicated data type (CRDT) .....	6
3.2. Different sequence CRDT algorithms: .....	6
3.3. Pseudo code for our approach:.....	7
3. Implementation Details: .....	8
4. Technology Stack .....	9
5. Application Screenshot .....	10
6. Server logs .....	10
7. Analysis of Expected Performance .....	11
8. Design of performance testing: .....	12
9. Test results: .....	13
10. Software and design Challenges .....	13
11. Future Scope .....	13
12. Related work: .....	13
13. References : .....	14

## Introduction

Online collaborative editing is a concept, where group of users geographically dispersed over a network are simultaneously working with the same set of files. Computer Supported Cooperative Work is the way 10 people work in groups with enabling technologies of computer networking and associated software, hardware, services and techniques. Developing a collaborative system involves people from geographically different locations working on a common project. Collaboration can be achieved in many ways but the simplest form of collaboration is through file editing and sharing. That is, files are retrieved from the web server, edited by a group of people and then saved on the web server in an asynchronous mode.

### 1. Goals

To implement a real time conflict-free collaborative which will provide the features mentioned below:

1. A user's edits (Insert/delete/update) are applied immediately.
2. There is minimal delay between when user A edits a document and when user B sees A's edits. (will be ensured for multiple users not just two)
3. Conflicts are merged in an intuitive and reasonable way
4. Provides text only chat functionalities so that users can chat with one another.

### 2. Distributed system challenges

**2.1. Communication Latency:** The complexity of real-time collaborative editing solutions stems from communication latency. In theory, if communication were instantaneous, then creating a real-time collaborative editor would be no more difficult than creating a single-user editor, because a document could be edited using an algorithm similar to the following:

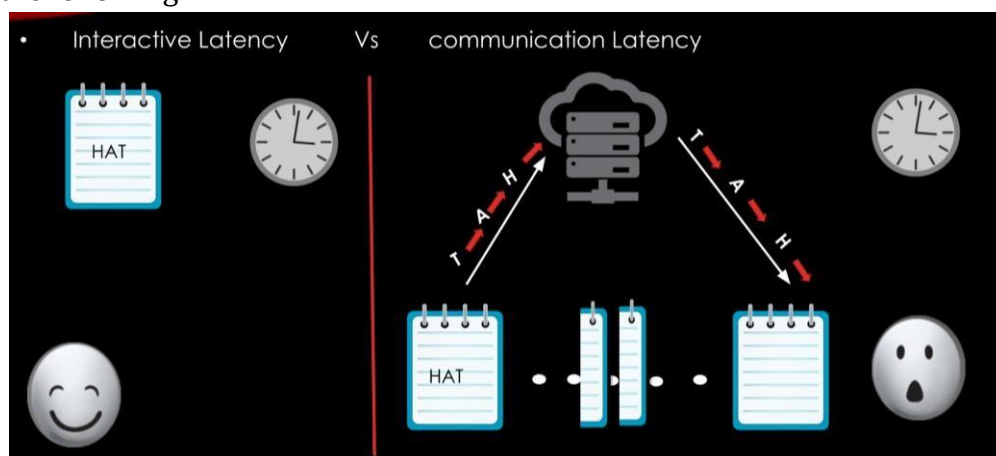


Figure 1: Communication Latency challenge

- Request an 'edit document' token from the server
- Wait until the server says it's our turn to edit the document
- Tell the server how to edit the document
- Release the 'edit document' token

Figure 1 illustrates this challenge. However, the speed of communication is limited by network latency. This creates a fundamental dilemma: users need their own edits incorporated into the document instantly, but if they are incorporated instantly, then because of communication latency, their edits must necessarily be inserted into different versions of the document.

**2.2. Concurrent Edits:** Handling concurrent editing in multi user environment gracefully is very challenging. The main challenge, as mentioned, with collaborative editing is the concurrency control [concurrent edits] to the document are not commutative. The concurrent edits if not dealt properly causes conflicts, for example if one user is inserting a character "C" at index 0 and at the same time another user is deleting the character "H" which is at index 0 then both these operation will cause inconsistency resulting in different results for different users.

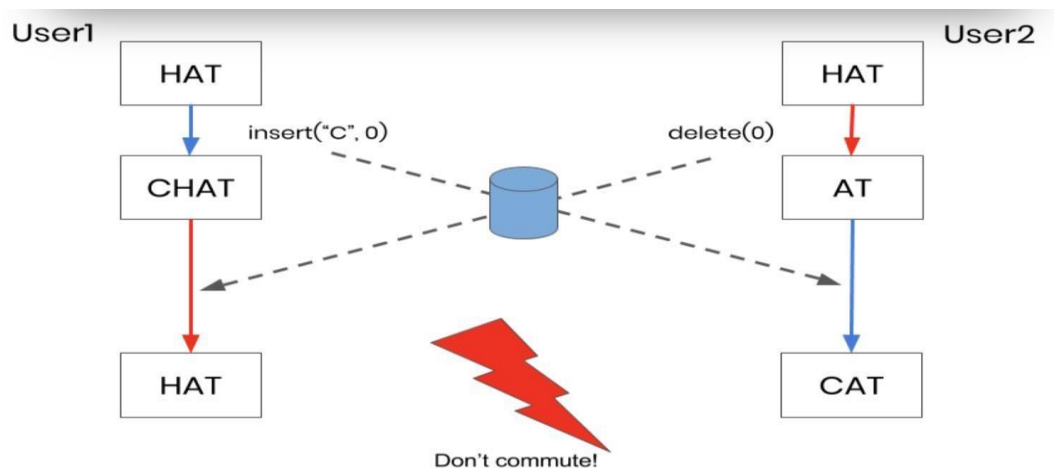


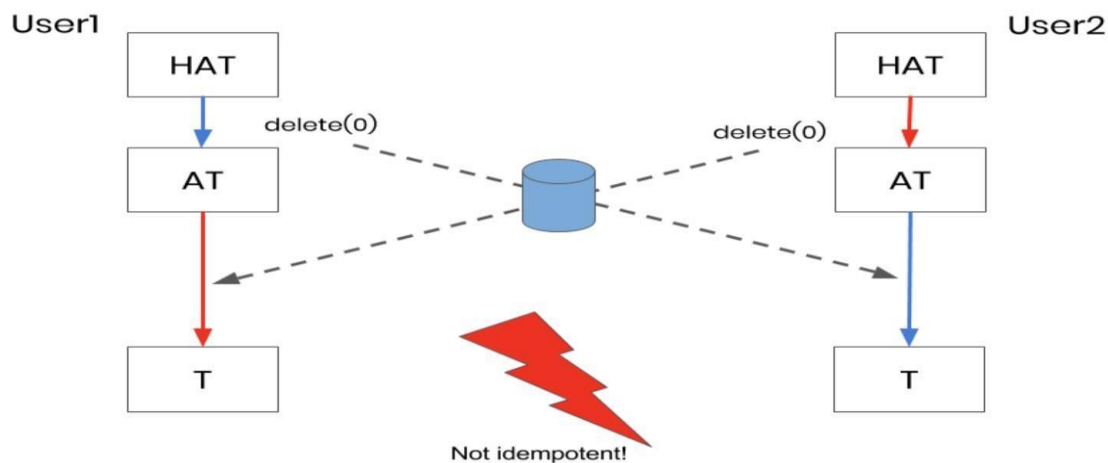
Figure 2: Challenge of communication in concurrent edits

This is shown very well in the figure 2.

### 2.3. Challenge of Idempotent:

Similarly, if both users are trying to delete the character "H" then there will be 2 delete operation but the delete should happen only once. So, our system should be capable of identifying duplicate deletes and should nullify one. This challenge is illustrated in figure 3.

- 2.4. Synchronization between all the clients : One of the main challenges of any distributed system is synchronization amongst all the clients. Our project ensures that at any given time all the client should be in sync.



- 2.5. **Maintaining the consistency and replication** : The data(text) that every client sees should be consistent.

### 3. How the challenges are addressed

#### 3.1. Conflict-free replicated data type (CRDT)

Conflict-free replicated data type (CRDT) is a data structure, which can be replicated across multiple clients connected to a network. Each replica can be updated independently and concurrently without co-ordination between the other replicas. Consequently, much of distributed computing centers on the challenge of how to prevent concurrent updates to replicated data. But another probable approach is optimistic replication, where all concurrent updates are permitted to go through, with inconsistencies may be created, and the outcomes are merged or "resolved" later. In this method, consistency between the replicas is eventually re-established via "merges" of conflicting replicas. While optimistic replication might not work in the general case, it turns out that there is a noteworthy and practically useful class of data structures, CRDTs, where it does work — where it is mathematically always possible to merge or resolve concurrent updates on different replicas of the data structure without conflicts. This makes CRDTs ideal for optimistic replication.

#### 3.2. Different sequence CRDT algorithms:

- LSEQ
- Lagoot
- Linklist-replicated
- Treedoc

We chose to go with the sequence CRDT algorithm which make use of Link-list to create the CRDT data structure.

### 3.3 Pseudo code for our approach:

We have implemented the pseudo code from the paper: A comprehensive study of Convergent and Commutative Replicated Data Types The pseudo code which we have implemented is as below:

```

1: payload set  $VA, VR, E$  ▷  $VA, VR$ : 2P-set of vertices;  $E$ : edges
2: ▷ Vertex = (atom, timestamp)
3:   let  $\perp = (\perp, -1)$ 
4:   let  $\neg = (\perp, 0)$ 
5:   initial  $\{\perp, \neg\}, \emptyset, \{(\perp, \neg)\}$  ▷ Initially, a single edge  $(\perp, \neg)$ 
6: query lookup (vertex  $v$ ) : boolean  $b$ 
7:   let  $b = (v \in VA \setminus VR)$ 
8: query before (vertex  $u$ , vertex  $v$ ) : boolean  $b$ 
9:   pre  $lookup(u) \wedge lookup(v)$ 
10:  let  $b = (\exists w_1, \dots, w_m \in VA : w_1 = u \wedge w_m = v \wedge \forall j : (w_j, w_{j+1}) \in E)$ 
11: query successor (vertex  $u$ ) : vertex  $v$ 
12:   pre  $lookup(u)$ 
13:   let  $v \in VA : (u, v) \in E$ 
14: query decompose (vertex  $u$ ) : atom  $a$ , timestamp  $t$ 
15:   let  $a, t : u = (a, t)$  ▷ Decompose  $u$  into atom, timestamp
16: update addRight (vertex  $u$ , atom  $a$ ) : vertex  $w$ 
17:   atSource ( $u, a$ ) :  $w$ 
18:   pre  $u \in VA \setminus (VR \cup \{\neg\})$  ▷ Graph precondition
19:   let  $t = now()$  ▷ Unique timestamp
20:   let  $w = (a, t)$ 
21:   downstream ( $u, w$ )
22:   pre  $u \in VA$  ▷ Graph precondition
23:   let  $a, t = decompose(w)$  ▷  $p = u$ 
24:    $l, r := u, successor(u)$ 
25:    $b := true$ 
26:   while  $b$  do ▷ Find an edge  $(l, r)$  within which to splice  $w$ 
27:     let  $a', t' = decompose(r)$ 
28:     if  $t < t'$  then ▷ Right position, wrong order
29:        $l, r := r, successor(r)$  ▷ Iterate
30:     else ▷  $r = \neg \vee t > t'$ 
31:        $E := E \setminus (l, r) \cup \{(l, w), (w, r)\}$ 
32:        $b := false$ 
33: update remove (vertex  $w$ )
34:   atSource ( $w$ )
35:   pre  $lookup(w)$  ▷ 2P-Set precondition
36:   downstream ( $w$ )
37:   pre  $addRight(\_, w)$  delivered ▷ 2P-Set precondition
38:    $VR := VR \cup \{w\}$ 

```

### 3. Implementation Details:

Our approaches for implementing CRDT are as follows.

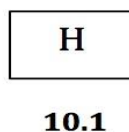
(i) We had unique global id naming system for each character. We used the timestamp of the server clock as the unique id's to the character entered by the user. But the problem with this approach failed if two or more users entered the characters at the same time or if the users were concurrent.

(ii) We made use of the Id's of the user who inserted the character in combination with the timestamp as an addition comparison field in order to decide the ordering of concurrent inserts/ deletes.

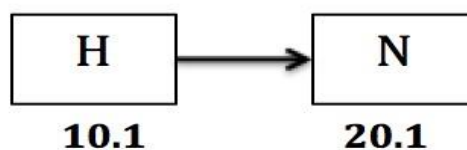
(iii) Each character entered by the user is treated as an object with properties associated with it.

Every character entered is treated as a node and a linked list data structures is used. Each node has a unique Id associated with it.

For example, assume that the editor is blank and User 1 is active. Now say User 1 enters letter "H" at timestamp 10. Since the editor is blank, the inserted letter would be placed at 0<sup>th</sup> index of the editor.



Now the same user enter letter "N" at timestamp 20. The previous character present in the editor would set its next node as the character that is being currently entered. The linking can be seen below.

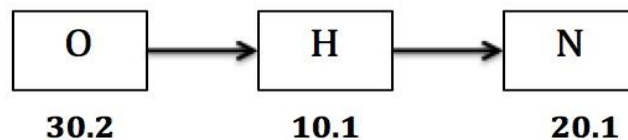


Now assume more users say User 2 and User 3 are active and are trying to enter letter "O" and "J" respectively at the same timestamp 30. Doth of these users are trying to insert at index 0.

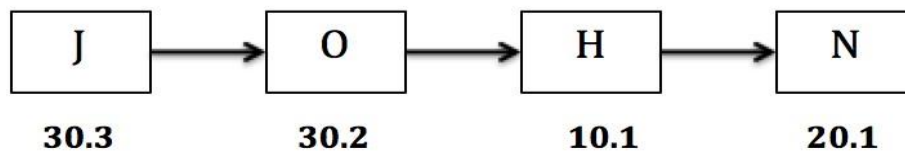




Since both the users are trying to insert during the same timestamp, the order of insertions can be decided by considering the order the user Id's. In this example, user 2 gets priority and letter "O" gets inserted first.



Now once letter "O" is inserted, we consider the next user that is user 3. Initially, the left variable for "O" was -1 which will be now updated with timestamp of "J".



Whenever there is an insert, the cursor's previous letter would be sent to an addright function. This function sets the new letter entered as the next letter of the sent letter. When there is no letter before the cursor (index 0), the -1 timestamp is sent which means that it is the very first character in the editor.

#### 4. Technology Stack

1. Node.js : Node.js is a platform built on top of chrome's JavaScript runtime that is capable of building scalable and fast network applications. It uses an event driven and nonblocking I/O model that is lightweight and efficient. It's a technology ideal for realtime data-intensive applications that run across distributed devices. Node's provides memory efficiency under high loads for each connection and prevents

deadlocks as no function of Node directly performs input/output operations. Node.js is an event-based model where a web server accepts the request and then passes this request to handlers. It then continues to service the next request. Once the previous request is completed, it goes back to the process queue and on reaching the front of queue the results are returned to the requesting client.

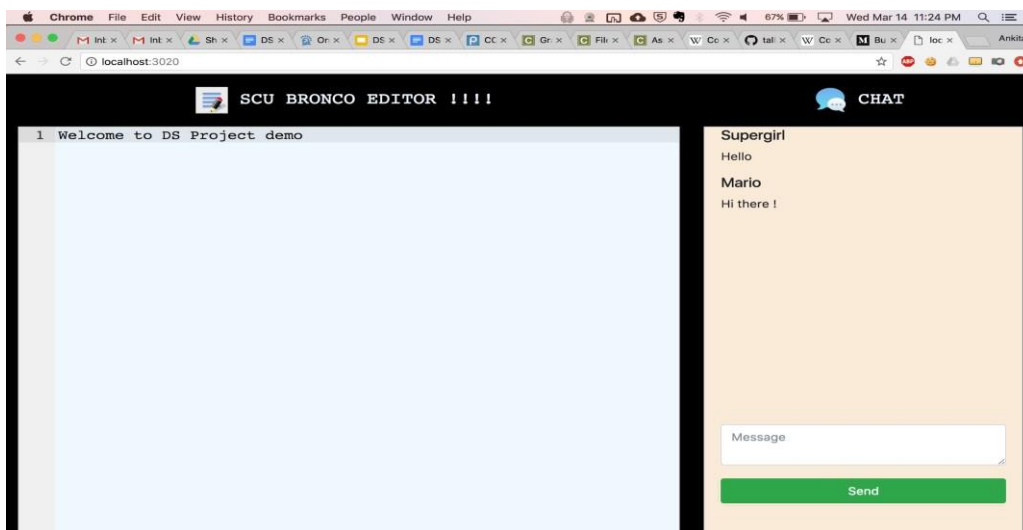
2. **Socket IO :** Socket IO is used to make real time applications on every browsers or mobile devices overcoming any differences between different transport mechanisms. This project uses Socket IO to create different client server connections.
3. **jQuery framework :** jQuery is a fast and efficient JavaScript library that provides event handling, Ajax interaction and simplified HTML traversing for rapid web development.
4. **User interface :** The user interface is developed using HTML , CSS

## Commands to run :

To run from terminal :

1. cd to the folder containing all the files and run the commands mentioned below
2. npm install --- this will install node in the respective folder
3. node server.js ---this will start the server
4. to start a client type localhost:3020 in your browser
5. you will see logs on the server window indicating the client is connected.

## 5. Application Screenshot



## 6. Server logs

Below are the server logs which indicate

- When a new client got connected,
- What and when the user typed(inserted) the character

- Where did the new character got inserted(you can see that in the Prev Node) ● It also gives logs for the chat.

```

Well done, now I am listening on 3020
connection - assigning id 1
Socket is connected...
connection - assigning id 2
Socket is connected...
Chat : user 2 : Hello
TextEdit : user 1 : Prev Node Id: -1 : Node Id: 1 : addRight : W
TextEdit : user 1 : Prev Node Id: 1 : Node Id: 65537 : addRight : e
TextEdit : user 1 : Prev Node Id: 65537 : Node Id: 131073 : addRight : l
TextEdit : user 1 : Prev Node Id: 131073 : Node Id: 196609 : addRight : c
TextEdit : user 1 : Prev Node Id: 196609 : Node Id: 262145 : addRight : o
TextEdit : user 1 : Prev Node Id: 262145 : Node Id: 327681 : addRight : m
TextEdit : user 1 : Prev Node Id: 327681 : Node Id: 393217 : addRight : e
TextEdit : user 1 : Prev Node Id: 393217 : Node Id: 458753 : addRight : t
TextEdit : user 1 : Prev Node Id: 458753 : Node Id: 524289 : addRight : o
TextEdit : user 1 : Prev Node Id: 524289 : Node Id: 589825 : addRight : t
TextEdit : user 1 : Prev Node Id: 589825 : Node Id: 655361 : addRight : D
TextEdit : user 1 : Prev Node Id: 655361 : Node Id: 720897 : addRight : S
TextEdit : user 1 : Prev Node Id: 720897 : Node Id: 786433 : addRight : S
TextEdit : user 1 : Prev Node Id: 786433 : Node Id: 851969 : addRight : P
TextEdit : user 1 : Prev Node Id: 851969 : Node Id: 917505 : addRight : r
TextEdit : user 1 : Prev Node Id: 917505 : Node Id: 983041 : addRight : o
TextEdit : user 1 : Prev Node Id: 983041 : Node Id: 1048577 : addRight : j
TextEdit : user 1 : Prev Node Id: 1048577 : Node Id: 1114113 : addRight : e
TextEdit : user 1 : Prev Node Id: 1114113 : Node Id: 1179649 : addRight : c
TextEdit : user 1 : Prev Node Id: 1179649 : Node Id: 1245185 : addRight : t
TextEdit : user 1 : Prev Node Id: 1245185 : Node Id: 1310721 : addRight : d
TextEdit : user 1 : Prev Node Id: 1310721 : Node Id: 1376257 : addRight : e
TextEdit : user 1 : Prev Node Id: 1376257 : Node Id: 1441793 : addRight : m
TextEdit : user 1 : Prev Node Id: 1441793 : Node Id: 1507329 : addRight : o
TextEdit : user 1 : Prev Node Id: 1507329 : Node Id: 1572865 : addRight : o
TextEdit : user 1 : Prev Node Id: 1572865 : Node Id: 1638401 : addRight : o
connection - assigning id 3
Socket is connected...
connection - assigning id 4
Socket is connected...
Chat : user 4 : Hi there !

```

## 7. Analysis of Expected Performance

The multiple concurrent client and the editing is tested with the selenium code by invoking the multiple (100) browsers at a time and by using the firebug plugin in the

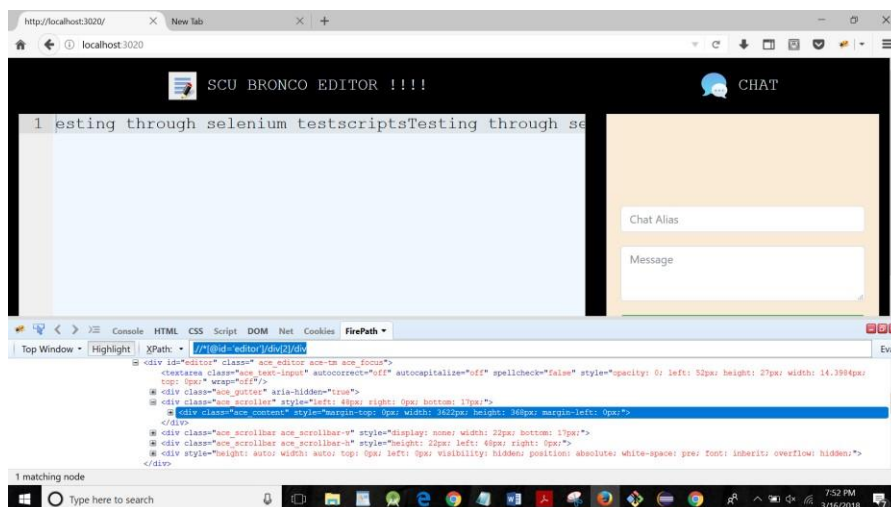
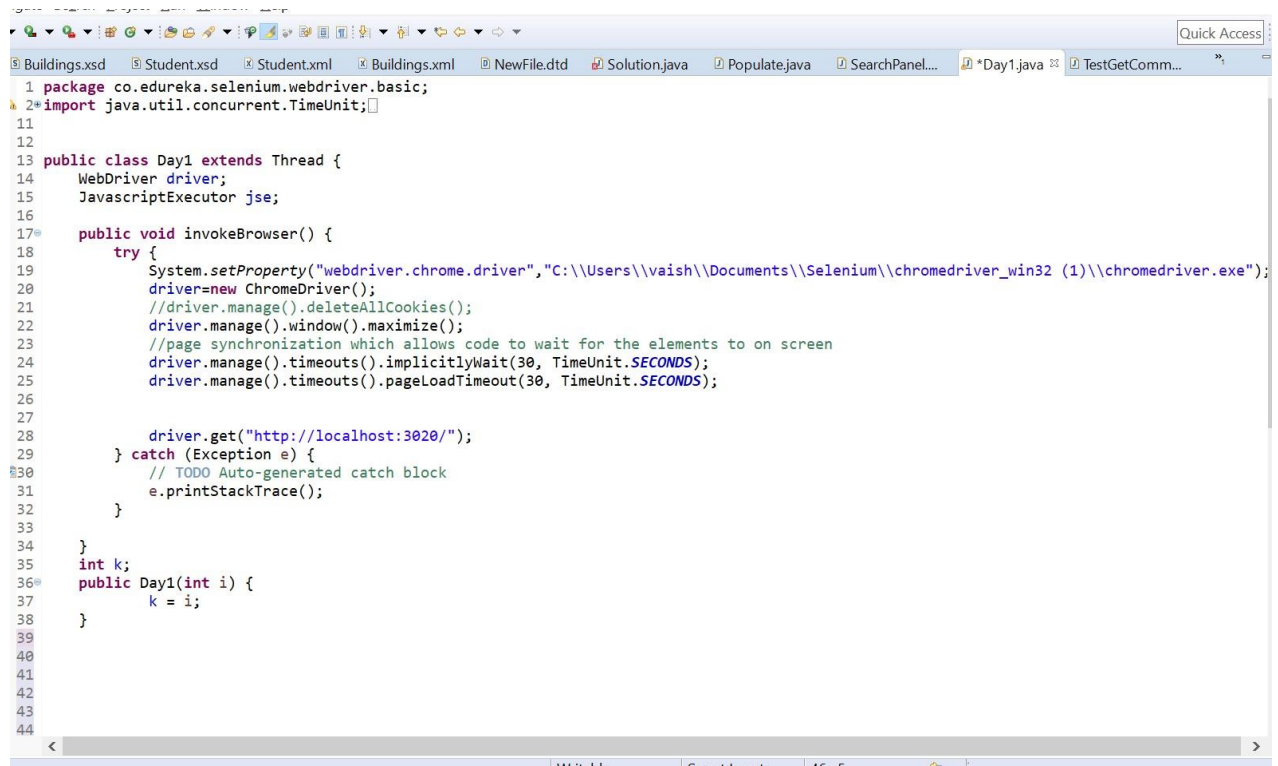


Figure4: Finding element on the browser for testing

Firefox fetched the xpath for the browser test-area for the editing. Below is the screenshot for the xpath for the same.

The keys are sent to the text-area through the selenium test scripts and observed the simultaneous editing for the 100 clients by spawning 100 threads.

## 8. Design of performance testing:



```
1 package co.edureka.selenium.webdriver.basic;
2 import java.util.concurrent.TimeUnit;
11
12
13 public class Day1 extends Thread {
14     WebDriver driver;
15     JavascriptExecutor jse;
16
17     public void invokeBrowser() {
18         try {
19             System.setProperty("webdriver.chrome.driver", "C:\\Users\\vaish\\Documents\\Selenium\\chromedriver_win32 (1)\\chromedriver.exe");
20             driver = new ChromeDriver();
21             //driver.manage().deleteAllCookies();
22             driver.manage().window().maximize();
23             //page synchronization which allows code to wait for the elements to on screen
24             driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
25             driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
26
27
28             driver.get("http://localhost:3020/");
29         } catch (Exception e) {
30             // TODO Auto-generated catch block
31             e.printStackTrace();
32         }
33
34     }
35     int k;
36     public Day1(int i) {
37         k = i;
38     }
39
40
41
42
43
44
```

```

37         k = i;
38     }
39
40
41
42
43 @Override
44 public void run()
45 {
46     invokeBrowser();
47     searchCourse();
48 }
49
50 public void searchCourse()
51 {
52     WebElement wb=driver.findElement(By.xpath("//*[id='editor']/div[2]/div"));
53     Actions actions = new Actions(driver);
54     actions.moveToElement(wb);
55     actions.click();
56     actions.sendKeys("Testing through selenium testscripts");
57     actions.build().perform();
58 }
59
60 public static void main(String[] args) throws InterruptedException {
61     // TODO Auto-generated method stub
62
63     for(int i=0;i<100;i++)
64     {
65         Day1 day1=new Day1(i);
66         day1.start();
67         day1.join(100);
68     }
69 }
70
71 }

```

## 9. Test results:

The invoked 100 clients were able to edit the text in the editor concurrently.

```

Select Git CMD - node server.js
Textedit : user 21 : Prev Node Id: 14876693 : Node Id: 14942229 : addRight : p
Textedit : user 21 : Prev Node Id: 14942229 : Node Id: 15007765 : addRight : t
Textedit : user 21 : Prev Node Id: 15007765 : Node Id: 15073301 : addRight : s
Textedit : user 23 : Prev Node Id: 15204375 : Node Id: 15466519 : addRight : i
Textedit : user 23 : Prev Node Id: 15466519 : Node Id: 15532055 : addRight : p
Textedit : user 23 : Prev Node Id: 15532055 : Node Id: 15597591 : addRight : t
Textedit : user 23 : Prev Node Id: 15597591 : Node Id: 15663127 : addRight : s
Textedit : user 27 : Prev Node Id: -1 : Node Id: 15400987 : addRight : e
Textedit : user 27 : Prev Node Id: 15400987 : Node Id: 15663131 : addRight : s
Textedit : user 27 : Prev Node Id: 15663131 : Node Id: 15728667 : addRight : t
Textedit : user 27 : Prev Node Id: 15728667 : Node Id: 15794203 : addRight : i
Textedit : user 27 : Prev Node Id: 15794203 : Node Id: 15859739 : addRight : n
Textedit : user 27 : Prev Node Id: 15859739 : Node Id: 15925275 : addRight : g
Textedit : user 27 : Prev Node Id: 15925275 : Node Id: 15990811 : addRight : g
Textedit : user 27 : Prev Node Id: 15990811 : Node Id: 16056347 : addRight : t
Textedit : user 27 : Prev Node Id: 16056347 : Node Id: 16121883 : addRight : h
Textedit : user 27 : Prev Node Id: 16121883 : Node Id: 16187419 : addRight : r
Textedit : user 27 : Prev Node Id: 16187419 : Node Id: 16252955 : addRight : o
Textedit : user 27 : Prev Node Id: 16252955 : Node Id: 16318491 : addRight : u
Textedit : user 27 : Prev Node Id: 16318491 : Node Id: 16384027 : addRight : g
Textedit : user 27 : Prev Node Id: 16384027 : Node Id: 16449563 : addRight : h
Textedit : user 27 : Prev Node Id: 16449563 : Node Id: 16515099 : addRight : i
Textedit : user 27 : Prev Node Id: 16515099 : Node Id: 16580635 : addRight : s
Textedit : user 27 : Prev Node Id: 16580635 : Node Id: 16646171 : addRight : e
Textedit : user 27 : Prev Node Id: 16646171 : Node Id: 16711707 : addRight : l
Textedit : user 27 : Prev Node Id: 16711707 : Node Id: 16777243 : addRight : e
Textedit : user 27 : Prev Node Id: 16777243 : Node Id: 16842779 : addRight : n
Textedit : user 27 : Prev Node Id: 16842779 : Node Id: 16908315 : addRight : i
Textedit : user 27 : Prev Node Id: 16908315 : Node Id: 16973851 : addRight : u
Textedit : user 27 : Prev Node Id: 16973851 : Node Id: 17039387 : addRight : m
Textedit : user 27 : Prev Node Id: 17039387 : Node Id: 17104923 : addRight : t
Textedit : user 27 : Prev Node Id: 17104923 : Node Id: 17170459 : addRight : e
Textedit : user 27 : Prev Node Id: 17170459 : Node Id: 17235995 : addRight : s
Textedit : user 27 : Prev Node Id: 17235995 : Node Id: 17301531 : addRight : t
Textedit : user 27 : Prev Node Id: 17301531 : Node Id: 17367067 : addRight : s
Textedit : user 27 : Prev Node Id: 17367067 : Node Id: 17432603 : addRight : c
Textedit : user 27 : Prev Node Id: 17432603 : Node Id: 17498139 : addRight : r
Textedit : user 27 : Prev Node Id: 17498139 : Node Id: 17563675 : addRight : i
Textedit : user 27 : Prev Node Id: 17563675 : Node Id: 17629211 : addRight : p
Textedit : user 27 : Prev Node Id: 17629211 : Node Id: 17694747 : addRight : t
Textedit : user 27 : Prev Node Id: 17694747 : Node Id: 17760283 : addRight : s
Textedit : user 27 : Prev Node Id: 17760283 : Node Id: 17825819 : addRight : s
connection - assigning id 29
Socket is connected...
connection - assigning id 30
Socket is connected...

```

## 10. Software and design Challenges

Each version of the node.js server is compatible with certain stable releases of connect middleware and Socket IO. All configurations needed to be compatible with each other.

## 11. Future Scope

In future work, we should be designing a more effective method to improve concurrency control by doing code optimization and try to improve performance, which would decrease users waiting time.

We would also want to host the application on cloud based platform to ensure availability and fault tolerance .Currently our project Bronco Editor supports single document but in future we would want to extend it to support multiple editor documents and include features like saving the document and having user accounts and also introduce a method to show which user is typing what by means of different cursor colors for different users.

## 12. Related work:

<https://hal.inria.fr/file/index/docid/555588/filename/techreport.pdf>

## 13. References :

- [https://en.wikipedia.org/wiki/Collaborative\\_real-time\\_editor](https://en.wikipedia.org/wiki/Collaborative_real-time_editor) ●
- [https://en.wikipedia.org/wiki/Conflict-free\\_replicated\\_data\\_type](https://en.wikipedia.org/wiki/Conflict-free_replicated_data_type) ●
- socket.io, Guillermo Rauch, gradebook learnboost labs.
- nodejs.org, Joyent Inc!