CS-553: Project Report

Sammed Sunil Admuthe (ssa180)
Jay Shivprasad Patil (jsp255)

GitHub Repository - https://github.com/SammedAdmuthe/collaborative-editor
Video Link - https://drive.google.com/file/d/1hijyQvoCDY_f72IJutL6gjdvIlBQV1w4/view?usp=sharing

Problem Statement:

The goal is to design a real-time collaborative editor to support editing for users. The real time editor would make sure read/write conflicts if any would be resolved in real-time thereby maintaining synchronization among clients.

Tech Stack:

We have used the following framework,modules and libraries in our application.

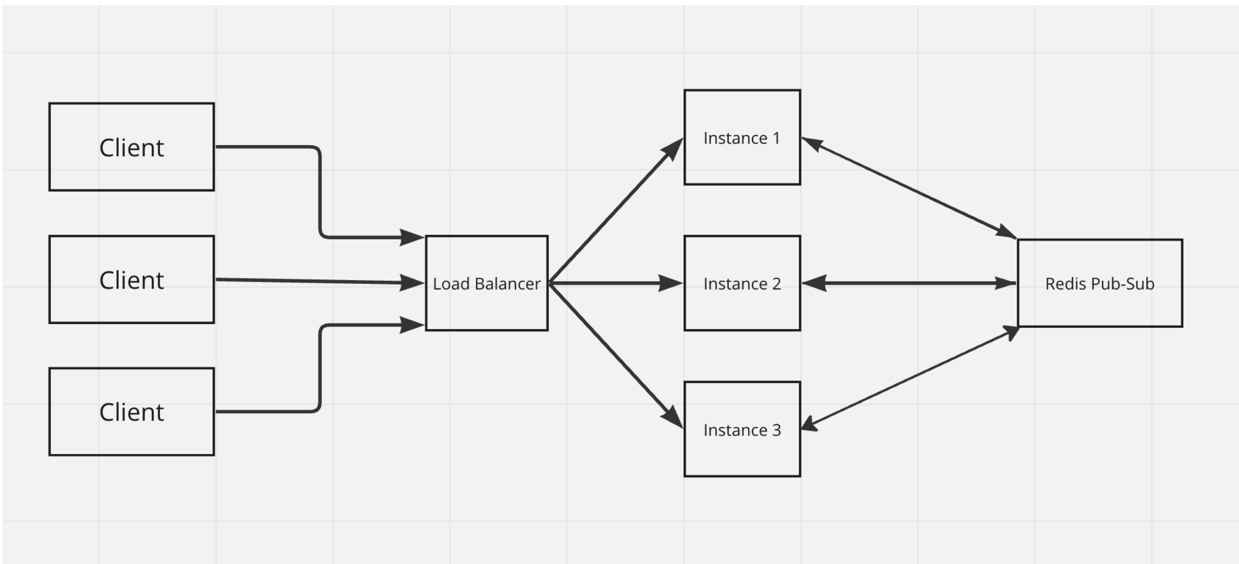
- NodeJS
- HAProxy
- Redis
- HTML/CSS
- Javascript
- Socket.IO
- Express
- CKEditor

Solution Implementation:

In this project, we have used operational transformation to support collaborative functionality. We believe the operational transformation would greatly enhance consistency and concurrency control for collaboration. We have used Redis as a publish subscribe message broker.

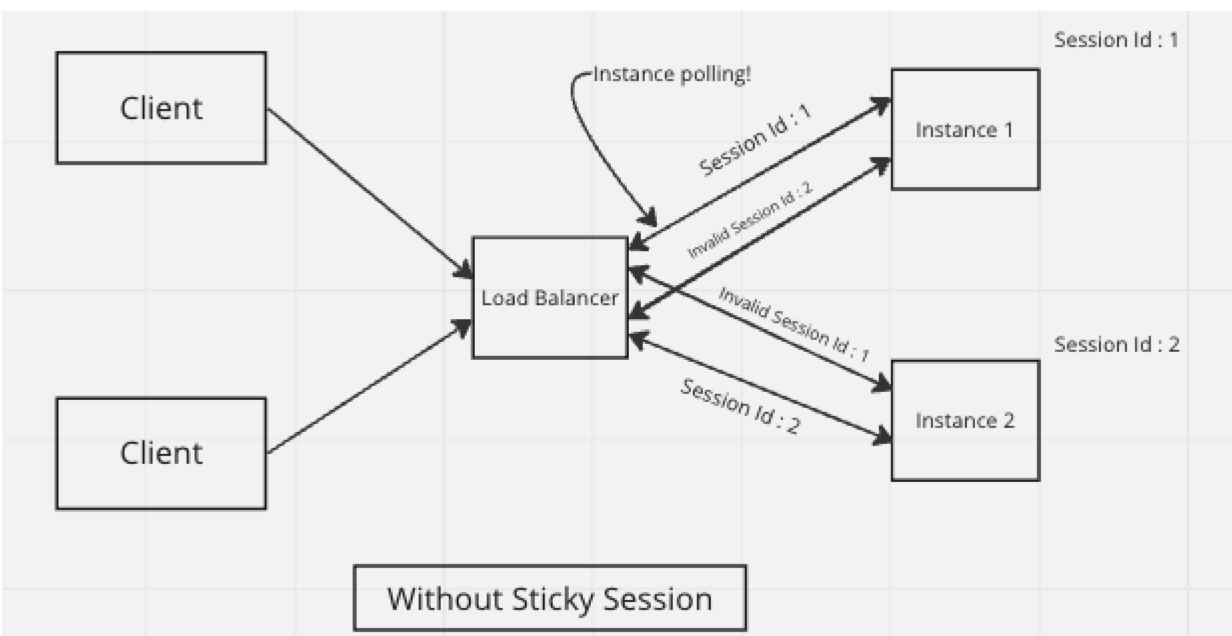
The designed collaborative editor is also scalable. The editor provides support for large traffic loads. The implementation involves socket synchronization. Inorder to distribute load across multiple instances we have used HAProxy like load balancer.

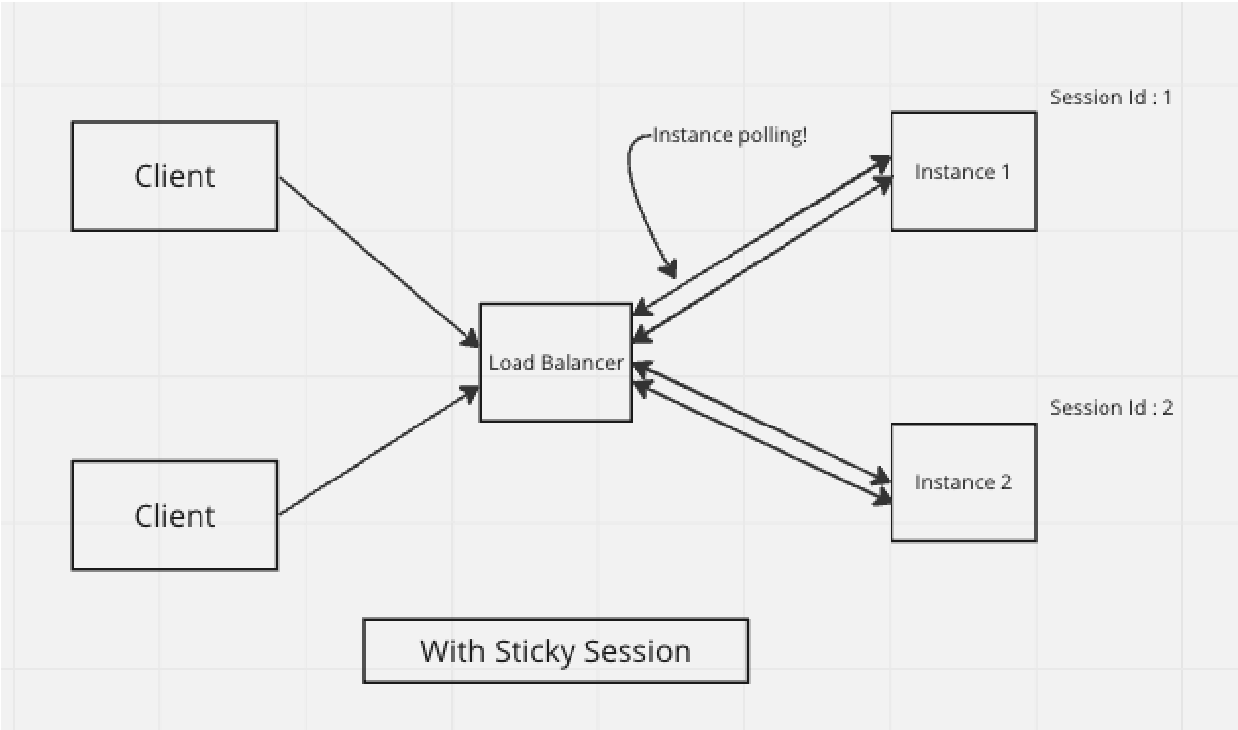
High Level architecture -



Sticky Session -

The project implementation involves the use of sticky sessions. The sticky session ensures that the load balancers consistently direct clients to the same server requests.





Why sticky sessions?

<u>Problems encountered with non sticky sessions</u>
While configuring load balancer, we encountered that the polling api used by Socket.IO contained a unique session identifier that belonged to the respective server. This made load balancer redirect the request to other instances during polling because of the round-robin strategy of execution. Thus, invalid session identifiers result in synchronization issues which ultimately ***<u>CRASHED</u>*** the application.

<u>Resolution</u>
We overcame this problem by configuring a unique cookie identifier for each server. This ensures that the client using a particular cookie gets redirected to the same server every time the request is being made. This eliminated the invalid cookie problem.

Setup and Implementation:

<u>HA Proxy</u>
HAProxy load balancer is responsible for distributing load across servers. Sticky session semantics have been configured in haproxy.cfg. Please refer to the config file shared in the GITHUB repository attached at the end of the report. Wherever the client hits <u>localhost:80</u>, the client basically hits load balancer which in turns redirects the request to other servers.

Instances

For the purpose of testing we have configured 3 servers on localhost but on different ports. Each server is capable of handling multiple requests from clients. Whenever the user hits localhost:80, the load balancer would redirect the client to the appropriate server based on the round robin mode of execution which is specified in the config file.

Redis

For running the server it is very important that redis is UP and RUNNING. Redis is running at the standard port - 6379. Redis publish-subscribe model ensures real-time synchronization among the clients. We have used the latest Redis - 7.0 version for our implementation.

CKEditor

CKEditor is an open-source web based text editor. This would serve as an editor for our application. Changes made to the editor would be conveyed to editors present on other server using Redis publish-subscribe model.

Note :- More information about setup could be found in README.md

Snapshots of 3 configured servers-

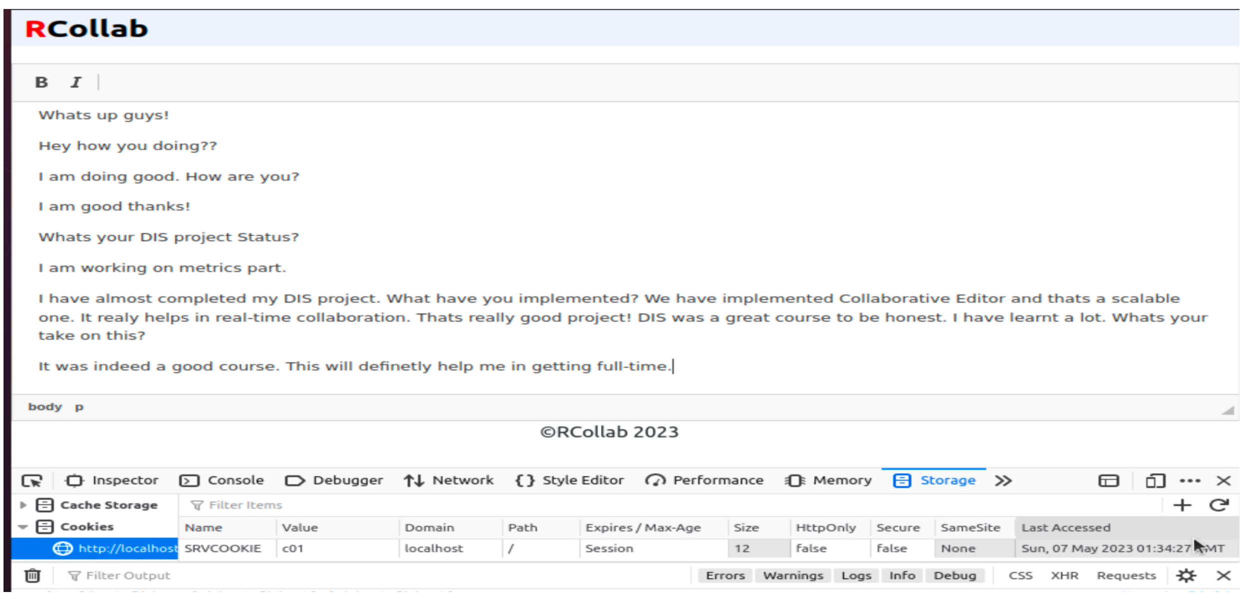


Fig 1:- Server 1 : SRVCOOKIE with value c01

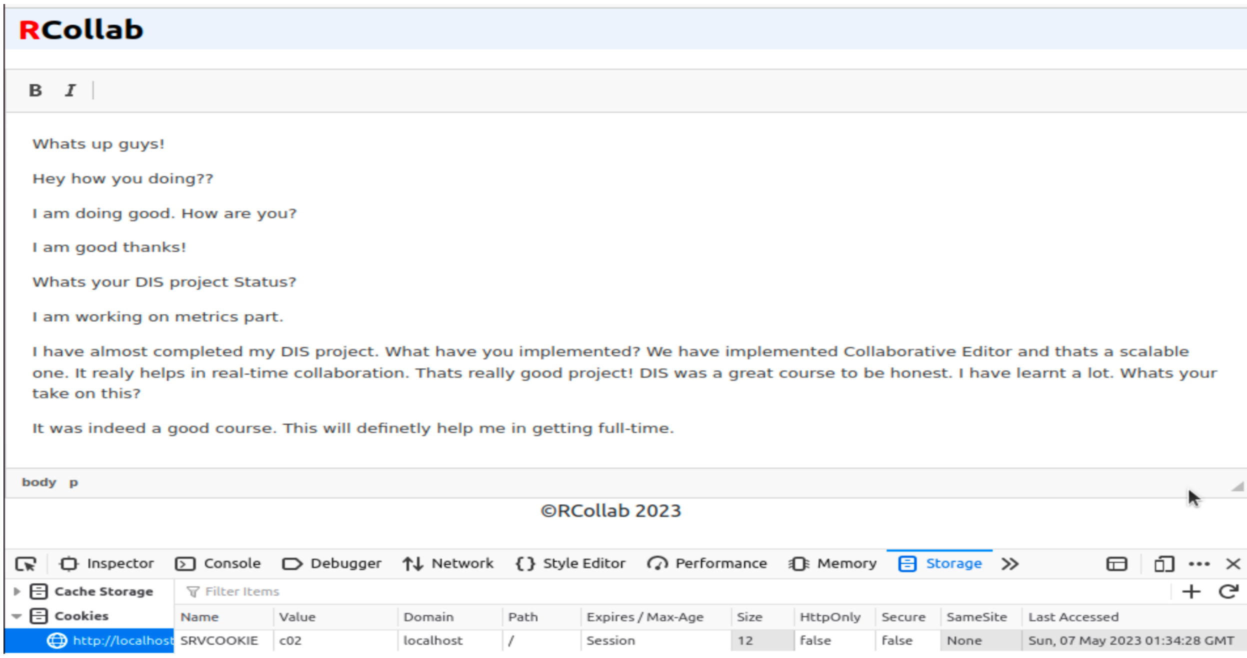


Fig 2:- Server 2 : SRVCOOKIE with value c02

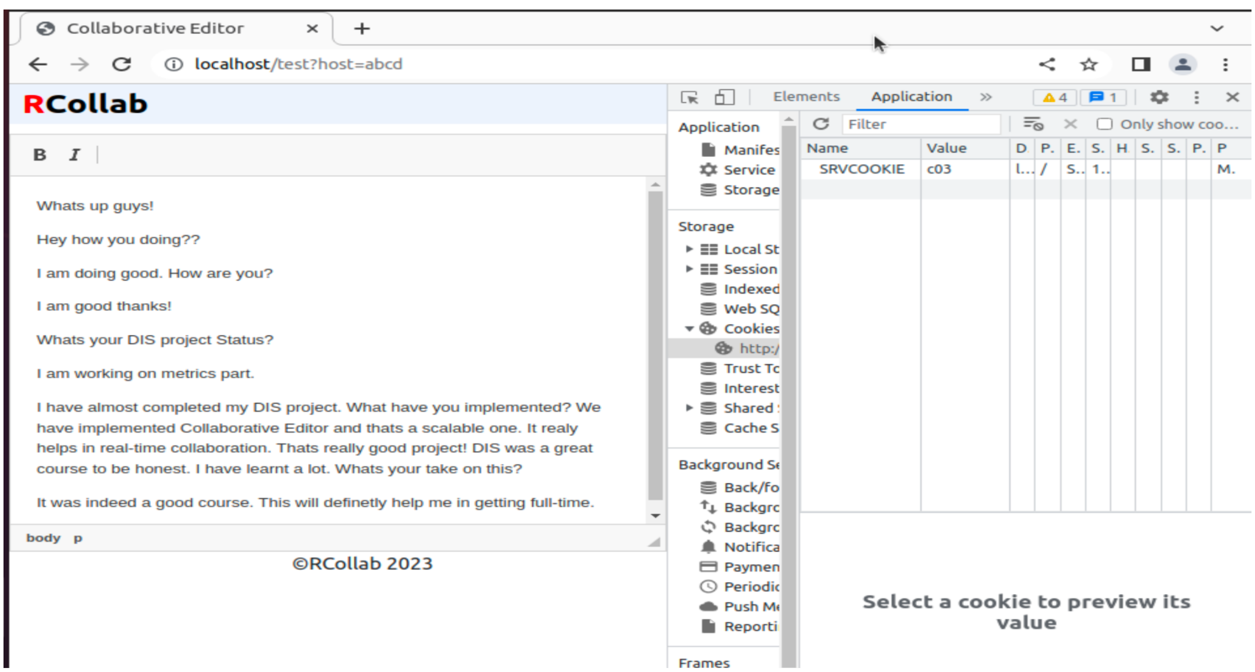


Fig 3:- Server 3 : SRVCOOKIE with value c03

Cookies ensure that once the connection is made to the server further polling requests are sticked to the same server for a given client-server connection.

Metrics and results:

We have a code in place to generate logs for the edit, insert and delete operational transformation event.


```
[{"type":"insert","index":0,"values":["<","p",">","L","e","t","s"," ","b","e","g","i","n"," ","t","e","s","t","i","n","g","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423252755
[{"type":"insert","index":21,"values":[" ","R","c","o","l","l","a","b"]}]|| Origin Server : -1 || Time stamp :  1683423255809
[{"type":"delete","index":0,"howMany":34}]|| Origin Server : -1 || Time stamp :  1683423261608
[{"type":"insert","index":0,"values":["<","p",">","W","h","a","t","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423421410
[{"type":"insert","index":7,"values":["s"," ","u","p"," ","g","u","y","s","!"]}]|| Origin Server : -1 || Time stamp :  1683423425654
[{"type":"insert","index":22,"values":["\n","<","p",">","H","e","y"," ","h","o","w"," ","y","o","u"," ","d","o","i","n","g","?","?","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423436944
[{"type":"insert","index":50,"values":["\n","<","p",">","I","<","/","p",">","\n"]}] || Origin Server : 1 || Time stamp :  1683423441518
[{"type":"insert","index":55,"values":[" ","a","m"]}] || Origin Server : 1 || Time stamp :  1683423443132
[{"type":"insert","index":58,"values":[" ","d","o","i","n","g"]}] || Origin Server : 1 || Time stamp :  1683423445514
[{"type":"insert","index":64,"values":[" ","g","o","o"]}] || Origin Server : 1 || Time stamp :  1683423447107
[{"type":"insert","index":68,"values":["d"]}] || Origin Server : 1 || Time stamp :  1683423448421
[{"type":"insert","index":69,"values":["."," ","H","o","w"," ","a","r","e"," ","y","o","u"]}] || Origin Server : 1 || Time stamp :  1683423453446
[{"type":"insert","index":82,"values":["?"]}] || Origin Server : 1 || Time stamp :  1683423457061
[{"type":"insert","index":88,"values":["\n","<","p",">","I"," ","a","m"," ","g","o","o","d","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423465302
[]|| Origin Server : -1 || Time stamp :  1683423466752
[{"type":"insert","index":101,"values":[" ","t","h","a","n","k","s","!"]}]|| Origin Server : -1 || Time stamp :  1683423470448
[{"type":"insert","index":114,"values":["\n","<","p",">","&","n","b","s","p",";","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423489153
[{"type":"insert","index":118,"values":["W","h","a","t"]}]|| Origin Server : -1 || Time stamp :  1683423492807
[{"type":"insert","index":122,"values":[" ","y","o","u","r"," ","p","r","o","j","e","c","t"]}]|| Origin Server : -1 || Time stamp :  1683423514355
[{"type":"insert","index":135,"values":[" "]},{"type":"delete","index":136,"howMany":3},{"type":"insert","index":137,"values":["t","a","t","u","s"]},{"type":"delete","index":142,"howMany":2}]|| Origin Server : -
stamp :  1683423521170
[{"type":"insert","index":122,"values":["s"]},{"type":"insert","index":129,"values":["D","I","S"," "]},{"type":"insert","index":141,"values":["S"]},{"type":"delete","index":142,"howMany":1},
{"type":"insert","index":147,"values":["?"]}]|| Origin Server : -1 || Time stamp :  1683423536523
[{"type":"insert","index":153,"values":["\n","<","p",">","I"," ","a","m"," ","w","o","r","k","i","n","g"," ","o","n"," ","m","e","t","r","i","c","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683
[{"type":"insert","index":179,"values":["s"," ","p","a","r","t","."]}]|| Origin Server : -1 || Time stamp :  1683423552387
[]|| Origin Server : -1 || Time stamp :  1683423555975
[{"type":"insert","index":191,"values":["\n","<","p",">","&","n","b","s","p",";","<","/","p",">","\n"]}]|| Origin Server : -1 || Time stamp :  1683423557408
```


Fig 4 :- Generated for logs for Operational Transformation.

<u>Eventual Consistency and conflict free writes/edits-</u>

We used these metrics to test the <u>availability of consistent data</u> across multiple servers. For this, we developed a simple test script that compares operational transformation changes for multiple servers.

Intuition :

There are two types of changes-

1. Changes originating from a server(Server Side Transmitted).
2. Changes propagated to a server from other servers(Client Side Received).

We compared the statistics for our 3 servers and found out that although data remains inconsistent for a small duration of time it is <u>eventually consistent</u> with all updates across servers. This ensures eventual consistency. We have designed a test case to test eventual consistency

Please refer to testScript.py in the gitHub repository for more information.

Timestamps for each Operational Transformation request-

# Insert Operation	# Delete Operation	Timestamp1	Timestamp2	Timestamp3	Origin Server
1	0	1683423252755	1683423252741	1683423252751	2
1	0	1683423255809	1683423255804	1683423255809	2
0	1	1683423261608	1683423261601	1683423261606	2
1	0	1683423421410	1683423421410	1683423421401	3
1	0	1683423425654	1683423425654	1683423425645	3
1	0	1683423436944	1683423436937	1683423436943	2
1	0	1683423441518	1683423441524	1683423441522	1
1	0	1683423443132	1683423443142	1683423443141	1
1	0	1683423445514	1683423445520	1683423445518	1
1	0	1683423447107	1683423447120	1683423447116	1
1	0	1683423448421	1683423448429	1683423448426	1
1	0	1683423453446	1683423453453	1683423453449	1
1	0	1683423457061	1683423457071	1683423457071	1
1	0	1683423465302	1683423465295	1683423465303	2
-1	-1	1683423466752	1683423466745	1683423466749	2
1	0	1683423470448	1683423470443	1683423470446	2
1	0	1683423489153	1683423489153	1683423489142	3
1	0	1683423492807	1683423492809	1683423492795	3
1	0	1683423514355	1683423514355	1683423514340	3
2	2	1683423521170	1683423521169	1683423521156	3
4	1	1683423536523	1683423536524	1683423536515	3
1	0	1683423548987	1683423548977	1683423548988	2
1	0	1683423552387	1683423552376	1683423552388	2
-1	-1	1683423555975	1683423555968	1683423555972	2
1	0	1683423557408	1683423557389	1683423557396	2
1	1	1683423558686	1683423558666	1683423558668	2
1	0	1683423569767	1683423569752	1683423569761	2
1	0	1683423581848	1683423581861	1683423581853	1
1	0	1683423600753	1683423600752	1683423600748	3
1	0	1683423606457	1683423606456	1683423606450	3
0	1	1683423607672	1683423607673	1683423607665	3
1	1	1683423612541	1683423612540	1683423612532	3
1	1	1683423618923	1683423618927	1683423618915	3
1	0	1683423622693	1683423622692	1683423622684	3
1	0	1683423625523	1683423625524	1683423625515	3
1	0	1683423627404	1683423627400	1683423627395	3
1	1	1683423630563	1683423630565	1683423630553	3
1	0	1683423640732	1683423640740	1683423640739	1
1	0	1683423643804	1683423643818	1683423643809	1
1	0	1683423678079	1683423678090	1683423678084	1
1	0	1683423682003	1683423682010	1683423682009	1
1	0	1683423690875	1683423690880	1683423690879	1
1	0	1683423713231	1683423713232	1683423713221	3
1	0	1683423719658	1683423719664	1683423719650	3
1	1	1683423723353	1683423723352	1683423723346	3
1	0	1683423725202	1683423725204	1683423725190	3

Table 1: Timestamps for 3 servers.

We are recording the logs for each insert delete operation on CKEditor. Table 1 indicates the captured logs and their corresponding response. One on change Operational Transformation record can contain multiple inserts and delete as indicated by #inserts and #deletes in each request.

Latency-

# Insert Operation	# Delete Operation	Latency for Server 1(in ms)	Latency for Server 2(in ms)	Latency for Server 3(in ms)
1	0	14	0	10
1	0	5	0	5
0	1	7	0	5
1	0	9	9	0
1	0	9	9	0
1	0	7	0	6
1	0	0	6	4
1	0	0	10	9
1	0	0	6	4
1	0	0	13	9
1	0	0	8	5
1	0	0	7	3
1	0	0	10	10
1	0	7	0	8
-1	-1	7	0	4
1	0	5	0	3
1	0	11	11	0
1	0	12	14	0
1	0	15	15	0
2	2	14	13	0
4	1	8	9	0
1	0	10	0	11
1	0	11	0	12
-1	-1	7	0	4
1	0	19	0	7
1	1	20	0	2
1	0	15	0	9
1	0	0	13	5
1	0	5	4	0
1	0	7	6	0
0	1	7	8	0
1	1	9	8	0
1	1	8	12	0
1	0	9	8	0
1	0	8	9	0
1	0	9	5	0
1	1	10	12	0
1	0	0	8	7
1	0	0	14	5
1	0	0	11	5
1	0	0	7	6
1	0	0	5	4
1	0	10	11	0
1	0	8	14	0
1	1	7	6	0
1	0	12	14	0

Table 2: Average delay for 3 servers.

Work load-

We did a simple load test for 3, 10 and 20 users.

No of Clients/Tabs	Server 1:Average Latency (in ms)	Server 2:Average Latency (in ms)	Server 3:Average Latency (in ms)
3	6.978	6.848	3.522
10	7.232	6.991	3.113
20	6.890	6.609	3.579

Reasons for the observed latencies -

We derived the average latency for each server using the table data described above. Server 3 has lower average latency(3.522) because most of the editing happens on the application hosted at server 3(Refer

Table 1). The latencies for Server 1 and Server 2 are almost identical because of the fact that these servers receive most of the content transmitted by Server 3.

Since the load is not large enough we could not establish a correlation between the number of clients and average latency of the server. But the intuition is that with an increase in the number of clients there would be an increase in server load and thus latency would increase at this point we would need to **vertically scale** our application.

Load balancer statistics-

HAProxy version 2.0.31-0ubuntu0.1, released 2023/03/22

Statistics Report for pid 129710

> General process information

pid = 129710 (process #1, nbproc = 1, nbthread = 2)
uptime = 0d 3h49m22s
system limits: memmax = unlimited; ulimit-n = 524288
maxsock = 524288; **maxconn =** 262122; **maxpipes =** 0
current conns = 6; current pipes = 0/0; conn rate = 0/sec; bit rate = 0.000 kbps
Running tasks: 1/24; idle = 100 %

- active UP
- active UP, going down
- active DOWN, going up
- active or backup DOWN
- active or backup DOWN for maintenance (MAINT)
- active or backup SOFT STOPPED for maintenance
- backup UP
- backup UP, going down
- backup DOWN, going up
- not checked

Note: "NOLB"/"DRAIN" = UP with load-balancing disabled.

Display option:

- Scope :
- Hide 'DOWN' servers
- Disable refresh
- Refresh now
- CSV export

External resources:

- Primary site
- Updates (v2.0)
- Online manual

api_gateway

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	0	-	0	0	262 122	0			0	0	0	0	0					OPEN								

stats

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	2	-	3	4	262 122	115			1 329 228	63 529 514	0	0	112					OPEN								

fe_main

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				0	3	-	3	8	262 122	164			103 050	72 581	0	0	0					OPEN								

dis_main

	Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
sammed1	0	0	-	0	6		1	3	-	22	1	1h55m	8 068	6 094		0		1	0	3	0	3h37m UP	L4OK in 0ms	1	Y	-	3	1	11m38s	-
sammed2	0	0	-	0	7		1	2	-	24	1	1h55m	10 192	10 990		0		0	0	0	0	3h37m UP	L4OK in 0ms	1	Y	-	3	1	11m31s	-
sammed3	0	0	-	0	6		1	3	-	37	1	1h55m	17 564	24 999		0		1	0	3	0	3h37m UP	L4OK in 0ms	1	Y	-	3	1	11m31s	-
Backend	0	0		0	7		3	4	26 213	215	3	1h55m	103 050	72 581	0	0		140	0	6	0	3h37m UP		3	3	0		1	11m28s	

Inference -

We successfully designed and implemented a scalable Collaborative editor. Having analyzed the application logs we can conclude that Operational Transformation is an efficient algorithm that considerably reduces server latencies and helps maintain eventual consistency. We tried to achieve conflict free editing and dealt with non-sequential writes.

References -

Operational Transformation or How Google Docs Works - David Chu @CocoaHeads Taipei
https://www.youtube.com/watch?v=u2_yccaHbQk&ab_channel=CocoaheadsTaipei

Google Wave: Live collaborative editing
https://www.youtube.com/watch?v=3ykZYKCK7AM&ab_channel=Google

"MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor"
https://hal.inria.fr/hal-01655438/document

"Shared editing on the web: A classification of developer support libraries"
https://ieeexplore.ieee.org/abstract/document/6680014

"COE: A collaborative ontology editor based on a peer-to-peer framework"
https://www.sciencedirect.com/science/article/pii/S1474034605000364

"A Container Orchestration Development that Optimizes the Etherpad Collaborative Editing Tool through a Novel Management System"
https://www.mdpi.com/20799292/9/5/828/html

"Specification and Complexity of Collaborative Text Editing"
https://software.imdea.org/~gotsman/papers/editing-podc16.pdf

Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System
https://www.cs.princeton.edu/courses/archive/fall22/cos418/papers/bayou.pdf

Creative conflict resolution strategy
https://dl.acm.org/doi/abs/10.1145/2145204.2145413

Differential Sync
https://neil.fraser.name/writing/sync/