Analysis of Contemporary Methodologies For Near-Real Time Collaboration

by

Moin Ahmed Qidwai

1155160971

A report
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
The Chinese University of Hong Kong

December 2021

Supervisor: Dr. Hong Xu, Henry

Contents

1 Abstract

Near-real time (NRT) collaboration is the goal of many software applications, from collaborative text-editors like Google Docs to modelling applications like Autodesk Maya. In reality most applications could benefit from allowing users to collaborate effectively regardless of the problem domain that the application targets. As such the reason that so few of the mainstream software supports this functionality is generally the result of complexity accompanied with solutions to this problem. While there are a few different methodologies for supporting NRT collaboration, in this paper we shall investigate YJS, a popular library implementing the YATA approach based on CRDTs along with Operational Transformation, the approach at the core of Google Docs. We shall then present comparisons of the two aforementioned techniques, along with results of tests conducted in real world environments.

2 Collaboration Objectives

In order for a solution to be considered for NRT collaboration, it must be able to satisfy certain conditions or objectives.

2.1 Eventual Convergence

Eventual convergence dictates that if two collaborators receive the same set of operations in any order, the end result of those operations must be the same for both the collaborators. [2]

That is, If we have a algorithm A for merging operations into a list and two sets of operations S_1 and S_2 that observe the below relation.

$$\forall o : o \in S_1 \leftrightarrow o \in S_2 \tag{1}$$

Then the following must hold true, where $\mapsto$ represents an input symbol.

$$S_1 \mapsto A \equiv S_2 \mapsto A \tag{2}$$

2.2 Intention Preservation

The idea behind preservation of intention is simple. Any solution that aims to provide for NRT collaboration must ensure the result is in accordance with the intention of all collaborators [1].

2.3 Interleaving

Interleaving occurs if two or more collaborators insert multiple characters at the same index, upon integrating their insertions the result may have their inputs mixed [3].
Example: Collaborators C_1 and C_2 add "vious" and "cious" to "pre". If interleaving occurs the result may be "prevcioiouuss". A program that allows for NRT collaboration must ensure interleaving cannot occur.

3 YATA

The YATA (Yet Another Transformation approach) is the core specification underlying YJS. This specification consists of two main components, a doubly linked list and a set of rules that all operations must observe [1].

3.1 Data Representation

The doubly linked list representation used by YATA is in contrast to other algorithms. Another popular algorithm is the RGA, which utilizes a uni-directional linked list. The doubly linked list allows YATA to avoid interleaving at the start of the document or in prepend operations. As such it can cater for a wider range of operations and use cases than RGA by default. Though due to storing a pointer to the successor the data representation in YATA requires more memory.

$$Block_i = (id_i, origin_{left}, origin_{right}, deleted_i, value_i) \tag{3}$$

Equation 3 represents a single element in the linked list of YATA. The origins represent the pointers to predecessor and successor elements [7].

$$id_i = (replica_i, counter_i) \tag{4}$$

Equation 4 represents the identifier for a single block in the linked list. It consists of the Replica ID (or user id) and the operation counter.

The above representation ensures each block has a unique identifier and a total order.

3.2 Operations

The YATA specification only outlines two types of operations: insertion and deletion. A combination of these operations can also lead to many others, for example the update operation.

3.2.1 Insertion

$$Operation_k = (id_k, origin_k, left_k, right_k, deleted_k, value_k) \tag{5}$$

Equation 5 represents the insertion operation with counter (k) [1]. The **origin** represents the predecessor for the block at the time of creation. The **left** and **right** represent the predecessor and successor respectively after the operation has been merged into the linked list. The **deleted** flag indicates if the block representing the operation has been marked for deletion. The **value** is the actual content that is to be inserted.

3.2.2 Deletion

The deletion operation is simply represented by setting the deleted flag of the insertion operation to **true** [5].

$$Operation_k = (id_k, origin_k, left_k, right_k, true, value_k) \tag{6}$$

3.2.3 Operation Ordering

Every operation block has a total order in the list defined by the natural predecessor relation $<$ [1].

$$O_1 < O_2 \leftrightarrow O_1 is\ a\ predecessor\ of\ O_2 \tag{7}$$

$$O_1 \leq O_2 \leftrightarrow O_1 < O_2 \vee O_1 \equiv O_2 \tag{8}$$

Given the above predecessor relation and insert operation we can represent an insertion between two operations $O_i\ and\ O_j$ as shown below.

$$Operation_{new} = (id_{new}, O_i, O_i, O_j, false, value_{new}) \tag{9}$$

In equation 9 the following relation must hold $O_i\ <\ Operation_{new}\ <\ O_j$.

As one may notice the origin and the left operation are the same in the above equation as this represents the operation at the time of it's creations. The origin for the operation is set at the time of the operation creation and does not change thereafter. The left pointer may change during the merge process of operations created by different replicas, if there are conflicts.

3.3 Rules of Conflict Resolution

As mentioned earlier in this paper YATA consists of certain rules that must be observed by operations specially in cases of conflicts [1]. These rules are the cornerstone of the YATA approach as they ensure **eventual convergence** and **intention preservation**.

3.3.1 Conflicting insertions

Insertion operations (O_a, O_b, ...) are in conflict if all of them are to be inserted between O_i and O_j. In the above example, if $Operation_{new}$ is to be integrated in the list of operations $L = [O_i, c_a, c_b, c_c ... O_j]$, then $Operation_{new}$ is in conflict with $[c_a, c_b, c_c, ...]$. The rules resolve these conflicts by calculating the index k for the new insertion. If the rules are observed by all collaborators then each of them calculates the same index [1]. Each rule can be illustrated as a predecessor relation $<_r$. As such if we observe these rules for $Operation_{new}$ then we integrate it between c_i and c_j where $\forall r : c_i \ <_r \ Operation_{new} \ <_r \ c_j$.

3.3.2 Rule One

The first rule dictates that for two conflicting insertions I_a and I_b that have different origins O_a and O_b respectively, the connection between I_a and O_a must not be intersected by the connection between I_b and O_b, or vice versa. The only instances where the above holds true is illustrated by the below ordered sets (and their opposites, which one can get by swapping the indexes a and b).

$$[O_a, O_b, I_b, I_a] \tag{10}$$

$$[O_a, I_a, O_b, I_b] \tag{11}$$

Set 10 represents the case where a operation and it's origin are inserted in between of the other operation and it's origin. Set 11 represents the case where a operation and it's origin are inserted after the other operation and it's origin.

Rule one then can be succinctly illustrated by the below equation.

$$O_a <_{r1} O_b \leftrightarrow O_a < Origin_b \vee Origin_b \leq Origin_a \tag{12}$$

3.3.3 Rule Two

Rule two is the standard rule of transitivity and can be illustrated by the below equation.

$$O_a <_{r2} O_b \leftrightarrow \forall O : O_b <_{r2} O \rightarrow O_a \leq O \tag{13}$$

3.3.4 Rule Three

Rule three dictates that if two conflicting insertions have the same origin, then the insertion with the smaller creator ID is to the left. It can be represented by the below equation.

$$O_a <_{r3} O_b \leftrightarrow Origin_a \equiv Origin_b \rightarrow Creator_a < Creator_b \tag{14}$$

3.3.5 Total Order Function

If we combine all the three rules by conjunction and utilize them for insertions we get a total order on the insertion operations. This ensures both **eventual convergence** and **intention preservation**.

3.4 YJS

YJS is an implementation of the YATA specification, though it does couple it with delta-state based operations. In other words it only passes the specific operations that changed per integration, as opposed to the full document as per the YATA specification. This ensures the size of the messages remains small and hence the burden on the network resources is minimized.

The main disadvantage of YJS along with YATA is that when each character is represented as an operation as opposed to a single character, the overall document takes greater space. Though this is compensated with generally lower time complexity and the small size of the propagated messages.

4 Operational Transformation

At it's core Google Docs utilizes Operational Transformation to provide the ability for NRT to it's users. The idea behind Operational Transformation is quite old yet still actively used, it was pioneered by C. Ellis and S. Gibbs in 1989 [8]. It consists of two core ideas as outlined by Srijan Agarwal [9].

- The document state represented as S, is updated using different Operations $O_1(S), O_2(S), ...$ (such as insertion and deletion).
- Conflicts resulting from concurrent updates are resolved using a transform function $O_3 = T(O_1, O_2)$ that takes the two operations in conflict and returns a new operation that can be applied to preserve intention.

4.1 Transformation Function

The transformation function mentioned above can take one of two different forms. [10]

4.1.1 Inclusion Transformation

The inclusion transformation function denoted $IT(O_1, O_2) \rightarrow O_3$ takes two operations O_1, O_2 and returns O_3, which effectively applies operation O_1 as if O_2 is included.

As an example lets say we have a document state (S) = "ACEF" at time T and we receive two concurrent updates to this document state represented by the below operations.

$$O_1(S) \equiv Insert(S, 1, B). \tag{15}$$

$$O_2(S) \equiv Insert(S, 2, D). \tag{16}$$

Operation shown in 15 will add B after the character at position = 1 (A). Operation shown in 16 will add D after the character at position = 2 (C).

Now if we apply the first operation to the state, the new state is equal to NS = "ABCEF". Applying the second operation to this will provide us with a final state FS = "ABDCEF". Clearly we lost the intention of the users, the first user intended for B to be added between A and C, the second user intended for D to be added between C and E. Instead, D was added between B and C. In order to rectify this we will apply the transformation function $IT(O_2, O_1)$ after applying O_1 to S, which transforms O_2 to the below operation.

$$O_3(S) \equiv Insert(S, 3, D). \tag{17}$$

Operation shown in 17 will add D after the character at position = 3 (C).

In general for a pair of character-wise operations **Insert(S, P, C)** (Insert character C after position P in state S) and **Delete(S, P)** (Delete character at position P in state S), four IT functions, denoted as $T_{ii}, T_{id}, T_{di}, T_{dd}$, can be defined as follows (I represents insert operations and D is for deletions).

$$T_{ii}(I(P1, C1), I(P2, C2)) = \begin{cases} I(P1, C1) & \text{if } P1 < P2 \ OR \ U_g \\ I(P1 + 1, C1) & \text{otherwise} \end{cases} \tag{18}$$

U_g in equation 18 is equal to $P1 == P2 \ AND \ U1 > U2$, where U1 and U2 are user identifiers used to break the tie.

$$T_{id}(I(P1, C1), D(P2)) = \begin{cases} I(P1, C1) & \text{if } P1 \leq P2 \\ I(P1 - 1, C1) & \text{otherwise} \end{cases} \tag{19}$$

$$T_{di}(D(P1), I(P2, C2)) = \begin{cases} D(P1) & \text{if } P1 < P2 \\ D(P1 + 1) & \text{otherwise} \end{cases} \tag{20}$$

$$T_{dd}(D(P1), D(P2)) = \begin{cases} D(P1) & \text{if } P1 < P2 \\ D(P1 - 1) & \text{if } P1 > P2 \\ I & \text{otherwise} \end{cases} \tag{21}$$

I in equition 21 is the special identity operator, and it returns the state as is since the deletion had already occurred. It is used as a tie-breaker for the delete/delete pair.

4.1.2 Exclusion Transformation

The exclusion transformation function denoted $ET(O_1, O_2) \rightarrow O_3$ takes two operations O_1, O_2 and returns O_3, which effectively applies operation O_1 as if O_2 is excluded.

Lets revisit the example from the previous section with the final document state F(S) = "ABCDEF". The operations are shown in 15 and 16.

In this case if we apply transformation function $ET(O_2, O_1)$ after applying O_1 to S, it will transform O_2 to the below operation.

$$O_3(S) \equiv Insert(S, 2, D). \tag{22}$$

Operation shown in 22 will add D after the character at position = 2 (B).

Exclusion transformation is useful when we wish to perform undo of operations. In the above O_3 we are simply applying O_2 as if we had undone O_1.

Generally if we have three operations O_a, O_b, O_c and we were to undo O_a, we would need to apply the following transformations to the original state S to get the final state.

$$O_{b\_final}(S) \equiv ET(IT(O_b, O_c), O_a) \tag{23}$$

$$O_{c\_final}(S) \equiv ET(IT(O_c, O_b), O_a) \tag{24}$$

4.2 The Undo Procedure

The undo algorithm must satisfy the following conditions [10].

- Undoing an operation O should transform the original state S into the final state FS, such that FS is the result of applying all operations besides O to S.
- Undoing all operations applied to S should bring the state back to S.

Formally we can represent the above conditions as below, given initial state S, operations O_a, O_b applied to it to get final state FS.

$$Undo(O_a) \rightarrow ET(O_b, O_a)(S) \tag{25}$$

$$Undo(O_a, O_b) \equiv S \tag{26}$$

4.3 State Storage in OT

At it's core Operational Transformation does not dictate how or where one should store the document state. The state could be managed through a peer-2-peer or client-server architecture, we discuss the two approaches below.

4.3.1 Peer To Peer

In a peer to peer implementation of operational transformation, there is no single source of truth for the document state. Every peer maintains a local copy of the document state and broadcasts their operations to every other peer along with a state vector containing the local copy of state of every peer. The transformation of incoming operations is done at each peer's node rather than at a central server.

Advantages

- There is no single point of failure as this approach does not rely on a central server.
- Offline collaboration (local network connections) can be supported with greater ease.

Disadvantages

- As the operations and state need to be broadcasted to each and every peer, this approach places high strain on network resources.
- The storage requirements are high as each peer needs to store the state vector containing the number of changes of each peer along with their identifiers.
- The transformation algorithm can be a lot more complicated given the set of divergent possible states between different peers can be high, as such there is a lack of real world implementations support P2P.

4.3.2 Client-Server

In a client-server implementation of operational transformation, there is a single source of truth for the document state (the server). Every peer maintains a local copy of the document state and it applies the local operations to this copy without need of locking the state. Each peer caches the locally applied operations that have not yet been sent to the server and at appropriate intervals it sends the operations to the server. In some client-server algorithms the client does not wait for an acknowledgement from the server but in Google Docs, the client waits for the server's acknowledgement before sending further operations to it. In our discussion we will assume that the client does wait for acknowledgements from the server.

Advantages

- The locally applied operations are instantaneous. (though depending on the algorithm this may be true of P2P implementations as well).

- The strain on the storage and network resources is low.
- The complexity of the transformation algorithm is much lower since the client only needs to reconcile the local state with that of the server.

Disadvantages

- If the server crashes then the whole system breaks down and may even lead to data loss.
- Offline collaboration over a local network can be more difficult to implement as one or more of the peers needs to act as a server.

5 CRDT and OT Comparison

We have explored two different methodologies along with their implementations for Near-real time collaboration, CRDT (YATA or YJS) and Operational Transformation (Google Docs). Below we discuss some of the advantages and disadvantages of each of these approaches.

5.1 CRDT

We will present a comparison solely for YATA as our choice of CRDT specification as comparing numerous specifications is out of scope for this report.

5.1.1 Advantages

- There is no need to wait for acknowledgement from other participants/server, increasing performance.
- Computation and resolution of a large number of simultaneous conflicts with a relatively low processing footprint.
- It is peer to peer by default, hence sharing in all the advantages of P2P architecture [4].

5.1.2 Disadvantages

- CRDT enforces certain conditions on operations as discussed previously, as such not all operations are compatible with CRDT.
- The memory requirements for CRDTs can be high depending on the structure used as significant meta data needs to be attached to each character and user.
- Since CRDTs are data types, they need to be created for each type of data that our users interact with, hence leading to high initial complexity.

5.2 Operational Transformation

We will compare operational transformation with a central server as it's the most common real world implementation.

5.2.1 Advantages

- Any operation can be supported as long as it's transformations can be defined.
- As they are far more mature in the real world, they are currently highly optimized for popular applications.

5.2.2 Disadvantages

- As it requires a central server, it is susceptible to single point of failure.
- The requirement for acknowledgement reduces the performance for clients.
- The transformation functions can get quite complex and proving their validity in all scenarios is difficult.

6 Performance Analysis and Testing

In this section we will outline benchmark tests performed using different implementations of YATA and OT. These will be compared to shed light on the relative performance advantages and disadvantages mentioned above. The code for the tests can be viewed at `https://github.com/moin-qidwai/CUHK_5720`.

6.1 Testing VM Specifications

The tests have been performed on a cloud based compute instance from Google Cloud Platform. The specifications are as shown below:

- CPU: 2600 MHz Per Core (Quad Core)
- Memory: 16 GB

Benchmarks

In order to analyze and compare the performance of OT and CRDTs, we have created separate applications to benchmark ShareDB (a OT implementation) and YJS (a CRDT implementation). Both of these applications share the same structure, are written in javascript and executed using the same versions of node on the same cloud host. In order to make the comparison fair we are also utilizing the same algorithm for inserting characters to the document. The algorithm is described below in brief.

Scenario	Frequency	Users
A	2	50
B	2	100
C	10	50

6.2 Algorithm

Each client inserts a character at the head of the document at set intervals. The intervals are different for each scenario but uniform across both YJS and ShareDB for that scenario (see below for different scenarios). Each client is also assigned an ID, the ID is a unique numeric value from 0 to N, where N is the number of clients - 1. If we represent time in terms of the number of characters inserted and label it using t, then at time t we select the character to insert as follows.

$$Char(t, id) = \begin{cases} id \ mod \ 10 & \text{if } t \equiv id \\ alphabet[(id \ + \ index) \ mod \ 27] & \text{otherwise} \end{cases} \tag{27}$$

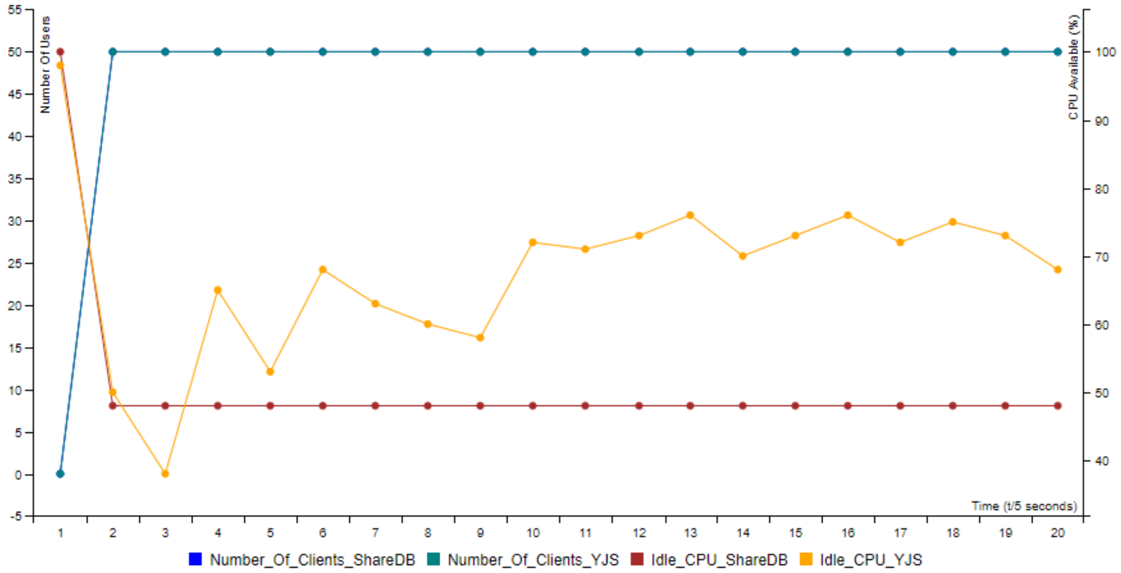
The alphabet list in equation (27) consists of the English alphabets with the space character at the end. Hence we have used modulo 27 to cycle through this list. This algorithm produces a very high conflict rate but also ensures each client produces a unique string based on their ID.

6.3 Scenarios

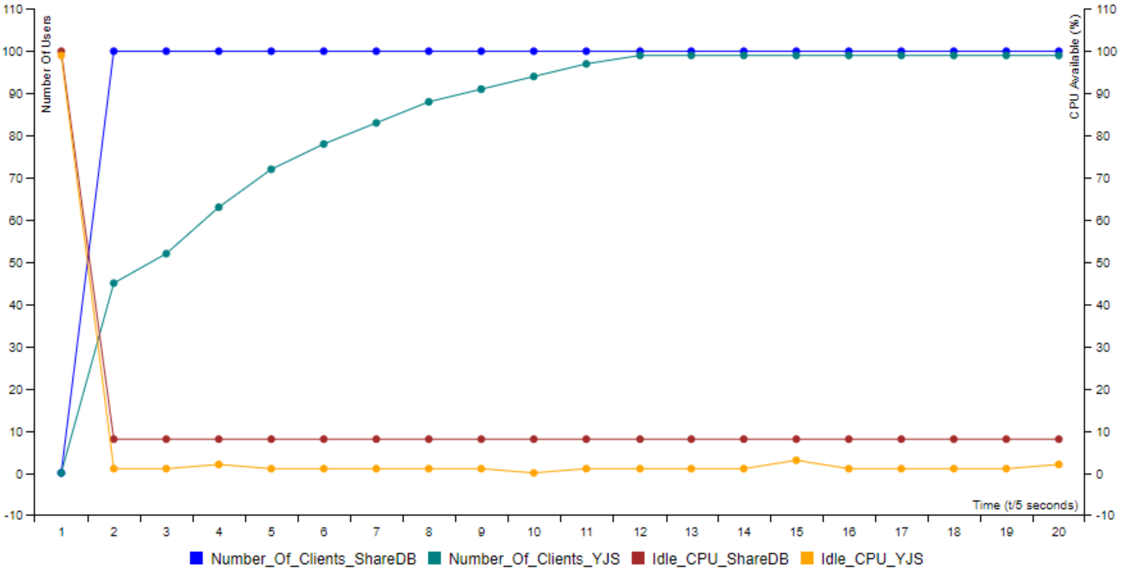
The below table shows the setup for each scenario, the frequency indicates the number of characters inserted per second and the users column shows the number of concurrent users.

6.4 Number Of Users

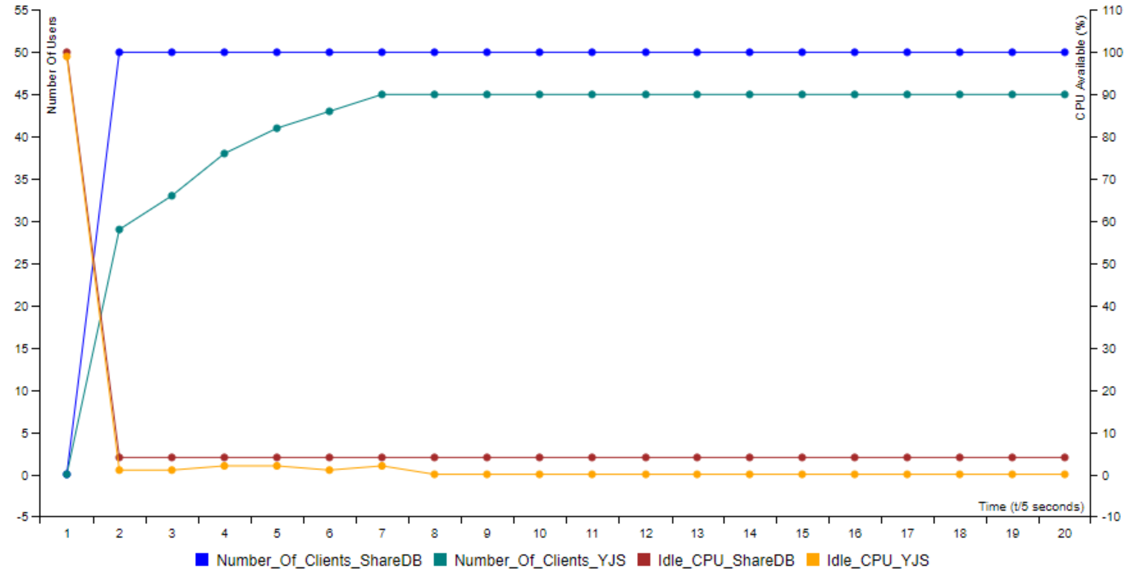
The below graphs indicate a lower performance for user/client connectivity in the case of YJS. This falls in line with the additional meta data requirement and distributed conflict resolution that exists for each client in YJS. In ShareDB the central server is the only process that maintains the information related to conflict resolution. This leads to a faster connectivity for clients in OT and a slightly lower total cpu usage of the whole network. It is worth noting that due to a higher total CPU utilization for YJS, the number of clients that were able to connect to the network was restricted.



Scenario A



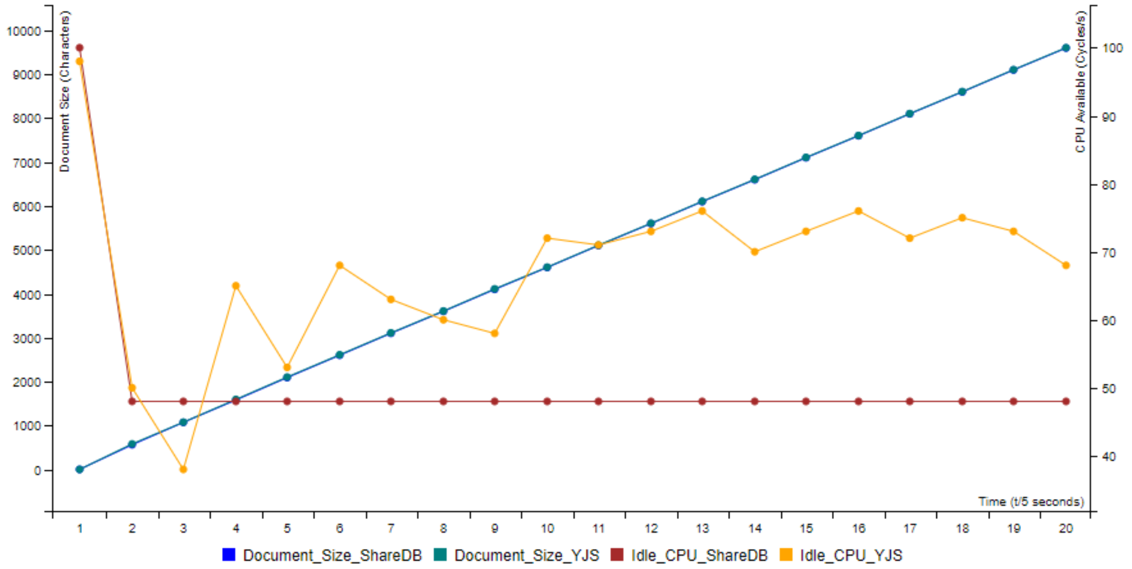
Scenario B



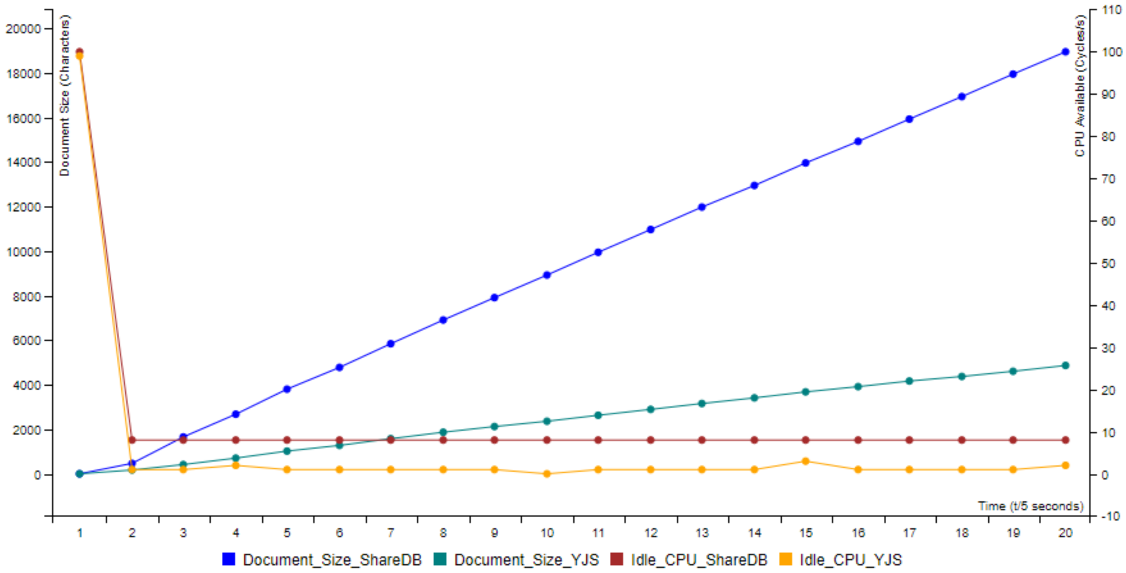
Scenario C

6.5 Document Size

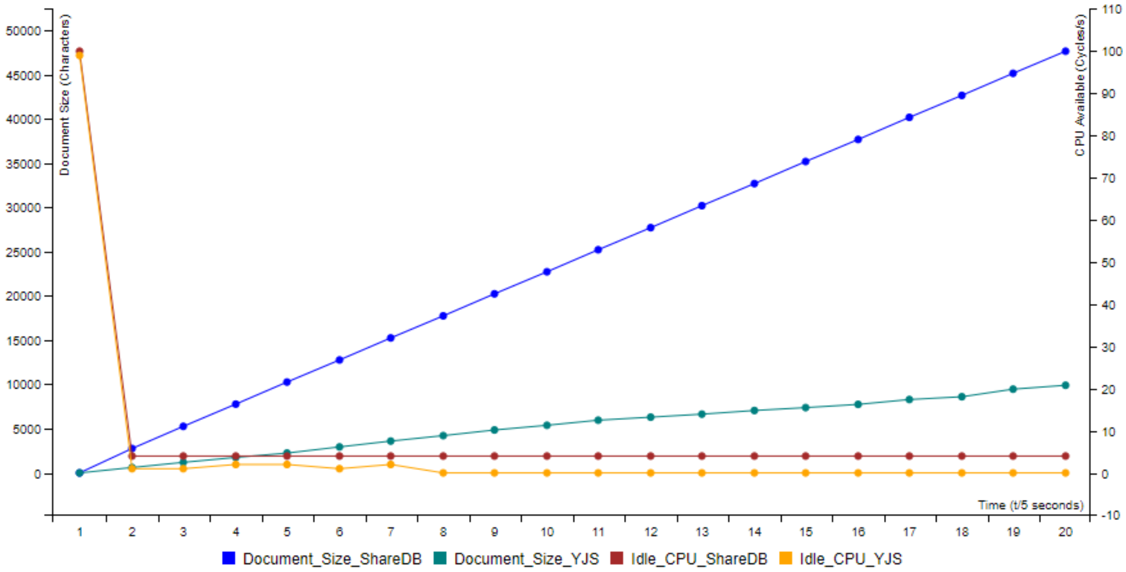
The performance for conflict resolution can be seen using the change in document size (pace of character insertions). Interestingly when the CPU is not fully utilized (Scenario A) the performance for both ShareDB and YJS is the same. Though once we fully utilize the CPU (Scenario C), the performance of ShareDB is better by a large margin.



Scenario A



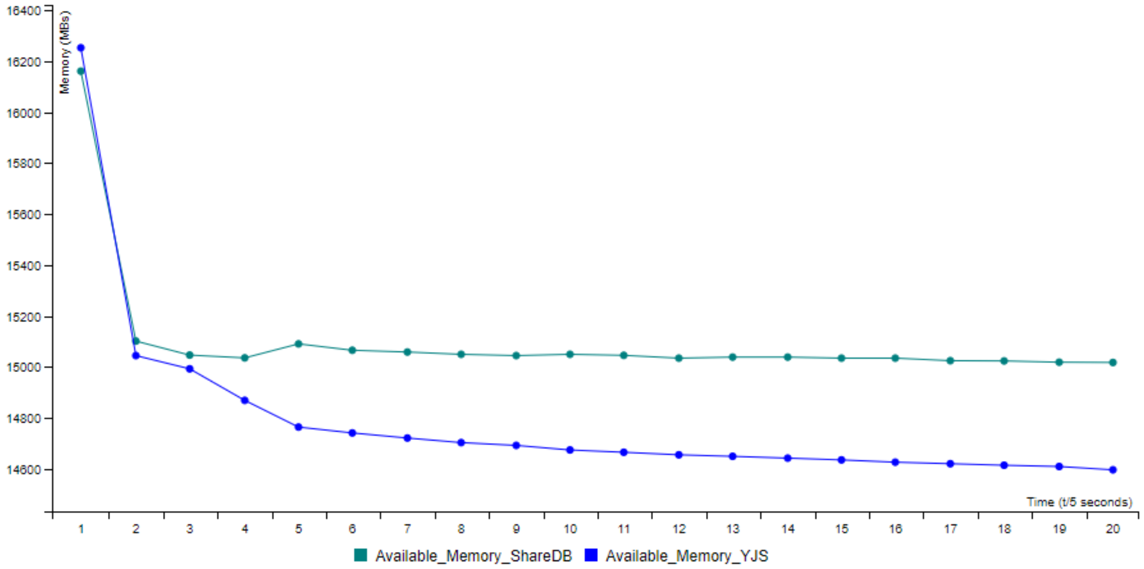
Scenario B



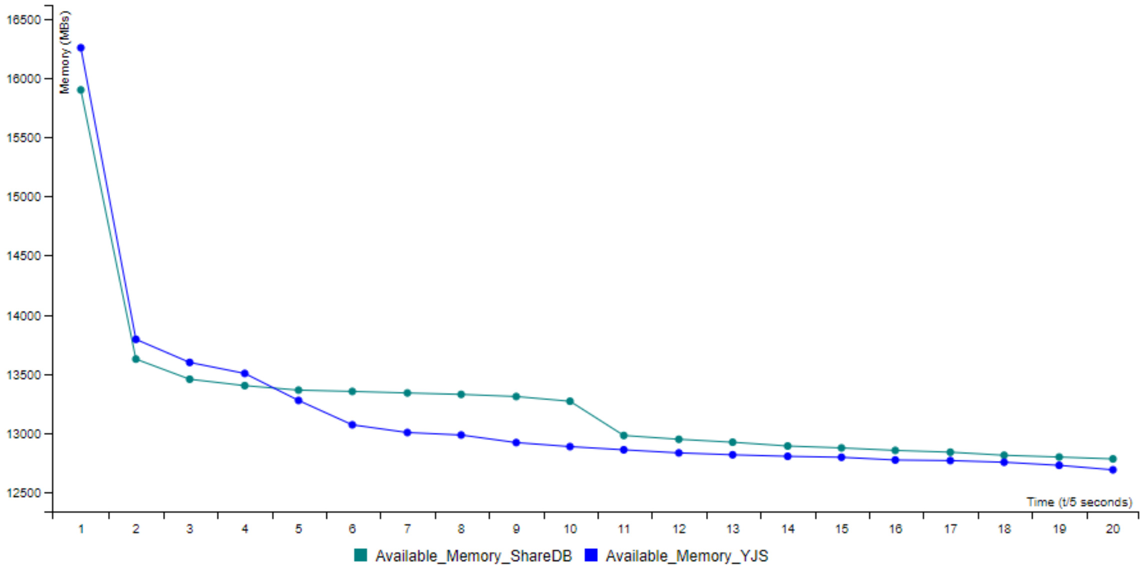
Scenario C

6.6 Memory Utilization

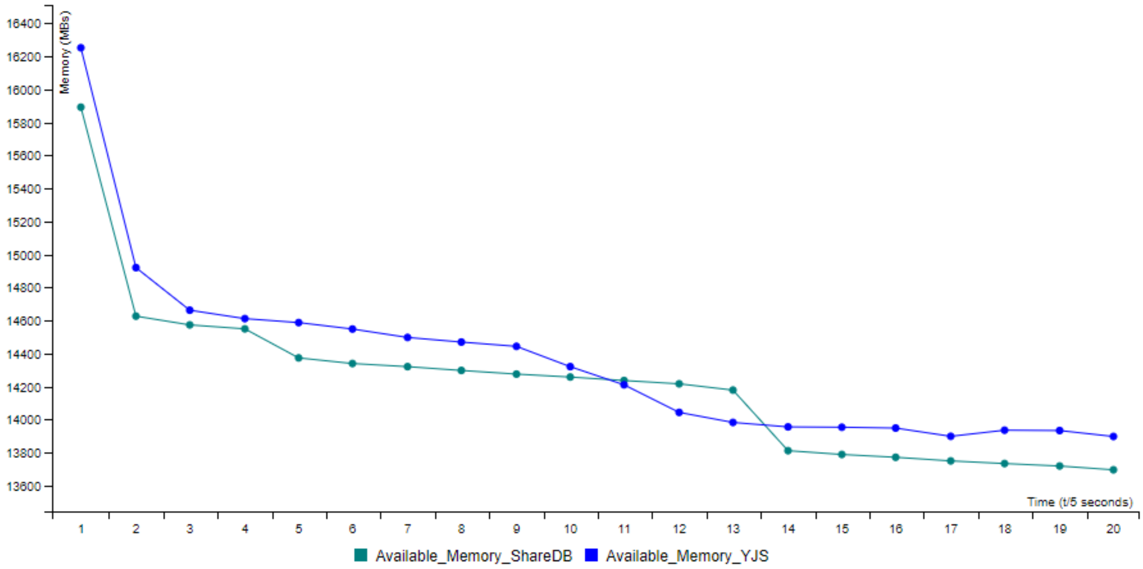
Memory utilization is quite similar for both YJS and ShareDB, though a bit higher for YJS. This is also due to the fact that in CRDT approaches each client must maintain additional meta data for the document contents. Though generally this is not a major issue as YJS is created for distributed utilization and hence the memory utilization is also distributed on several machines.



Scenario A



Scenario B

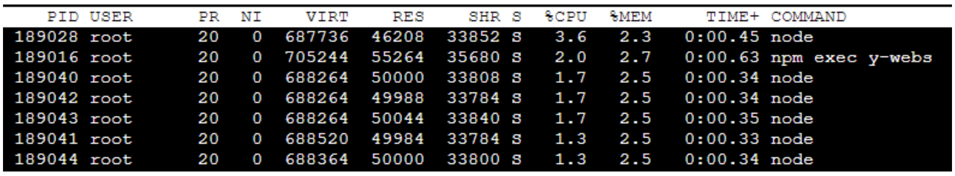


Scenario C

6.7 Process Snapshot

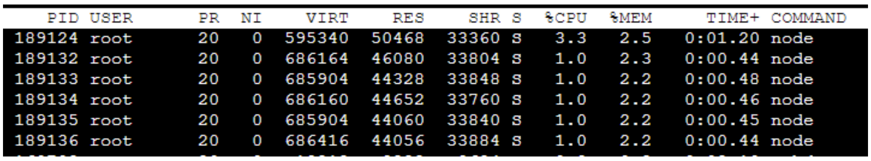
The below images show process information that verifies that YJS has a higher per client CPU utilization. The top node process for both YJS and ShareDB is the server. The five node processes below in the list are the clients. The average CPU utilization per client for YJS is higher than that of ShareDB.

6.7.1 YJS



```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
189028 root      20   0  687736  46208  33852 S   3.6   2.3   0:00.45 node
189016 root      20   0  705244  55264  35680 S   2.0   2.7   0:00.63 npm exec y-webs
189040 root      20   0  688264  50000  33808 S   1.7   2.5   0:00.34 node
189042 root      20   0  688264  49988  33784 S   1.7   2.5   0:00.34 node
189043 root      20   0  688264  50044  33840 S   1.7   2.5   0:00.35 node
189041 root      20   0  688520  49984  33784 S   1.3   2.5   0:00.33 node
189044 root      20   0  688364  50000  33800 S   1.3   2.5   0:00.34 node
```

6.7.2 ShareDB



```
   PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
189124 root      20   0  595340  50468  33360 S   3.3   2.5   0:01.20 node
189132 root      20   0  686164  46080  33804 S   1.0   2.3   0:00.44 node
189133 root      20   0  685904  44328  33848 S   1.0   2.2   0:00.48 node
189134 root      20   0  686160  44652  33760 S   1.0   2.2   0:00.46 node
189135 root      20   0  685904  44060  33840 S   1.0   2.2   0:00.45 node
189136 root      20   0  686416  44056  33884 S   1.0   2.2   0:00.44 node
```

6.8 Accuracy

Previously we have looked at the hardware resource utilization per client for both YJS and ShareDB. In such comparisons as noted ShareDB does perform slightly better than YJS. Now we zoom in to look at the accuracy of the actual text produced when inserting characters. This is so that we can ascertain the "quality" of the conflict resolution that each approach offers, rather than simply the "quantity". We define a accuracy metric based on a simpler version of our insertion algorithm illustrated by equation (28). Equations (29) to (31) illustrate the formula for determining accuracy.

$$Char(index, id) = alphabet[index\ mod\ 26] \tag{28}$$

$$Accuracy(text, clients) = 100 - ErrorRate(text, clients) \tag{29}$$

$$ErrorRate(text, clients) = (\sum Err(text[index], index, clients)) \div length(text) \tag{30}$$

$$Err(c, i, n) = \begin{cases} 1 & \text{if } alphabet[ceil(((i\ mod\ n * 26) + 1) \div n) - 1] \neq c \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

The indices in equations above start from 0 and alphabet is the list of alphabet a to z, in ascending order.

The below table shows the test runs and their accuracy for both YJS and ShareDB. Each of these tests were run in isolation on the same cloud host described above in the performance evaluation section. The number of clients for each test was 3.

Test Number	Accuracy (%) - ShareDB	Accuracy (%) - YJS
1	99.92	70.06
2	92.83	70.53
3	86.35	50.79
4	24.08	51.61
5	99.86	66.36
6	92.11	54.69
7	47.83	64.51
8	91.95	72.77
9	59.97	88.03
10	58.73	64.30

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
ShareDB	78.828	89.15	99.92	24.08	26.00
YJS	65.365	65.44	88.03	50.79	11.23

6.8.1 Concluding Remarks

It is clear from our testing and performance analysis that neither of the two approaches (CRDT or OT) perform well qualitatively. The accuracy metric that we measured in the previous section illustrates a mean error rate of atleast 22% with only three clients. As both approaches are fundamentally targeted towards collaboration, we can assume that three clients is likely a minimum number of clients in real world scenarios. In light of this we investigate another approach pioneered by Microsoft called Total Order Broadcast. This approach is implemented in a library called Fluid which is made publicly available by Microsoft for use. We will investigate this library and it's approach in the next section.

7 Total Order Broadcast

The fundamental principle behind total order broadcast is to ensure a consistent order to the operations being applied by different clients using a central service. When a client makes a change to the local state, that change is first sent to the central service which does three things [11]:

- Assigns a monotonically increasing sequence number to the operation; this is the "total order" part of total order broadcast.
- Broadcasts the operation to all other connected clients; this is the "broadcast" part of total order broadcast.
- Stores the operation's data

As each of the operation is ordered by the central service, the total order remains consistent for every client. The requirement for a central service makes this approach unfeasible as a P2P solution to collaborative editing similar to Operational Transformation. Let us first compare total order broadcast with the previously explored approaches of CRDT and OT.

7.1 Fluid

The only implementation of total order broadcast currently available is from Microsoft itself, the name of this framework is Fluid. The Fluid framework is currently in preview, and is also available through Azure. Though given the relative immaturity of Fluid it suffers from certain issues that have already been resolved in other libraries like ShareDB. When conducting our comparisons, we will assume Fluid to be in it's current state and hence assume those issues to be part of our comparison, though given the pace of development of this framework our comparison may be outdated by the time of distribution of this report.

7.2 Total Order Broadcast and CRDT Comparison

The advantages and disadvantages of Total Order Broadcast when compared to CRDT are outlined below.

7.2.1 Advantages

- Users are allowed to define their own custom merge strategies for conflict resolution, leading to high extensibility.
- Being backed by Microsoft, Fluid has extensive development resources and long term support.

7.2.2 Disadvantages

- As it requires a central server, it is susceptible to single point of failure.
- The requirement for acknowledgement reduces the performance for clients.
- The relative immaturity leads to poor performance when compared to CRDTs.

7.3 Total Order Broadcast and OT Comparison

The advantages when compared to OT are similar to the one's outlined above in our comparison to CRDT. The disadvantages of Total Order Broadcast when compared to OT are outlined below.

- ShareDB significantly optimizes the size of each communicated operation hence Fluid consumes more network resources.
- The relative immaturity leads to poor performance when compared to OTs.

7.4 Preliminary Performance Analysis

A basic integration of the fluid framework was tested using our formula for insertion as outlined in section 6.8. The performance was drastically poor even for just a few clients. Traversing the code of the framework few potential causes were identified as outlined below.

- Absence of inflight operational handling as all operations are sent regardless of server's acknowledgements.
- The raw textual representation of the operations transferred over the network.
- A lack of optimization around the websocket layer specifically for transferring Fluid Operations.

As all these features seem to have been developed in other frameworks (like ShareDB) over decades, it is unrealistic to expect Fluid to be able to deliver similar quality in a matter of a few years. As such Fluid was discarded as a potential solution to the poor accuracy we encountered during our testing of CRDT and OT. We then embarked upon adding a proprietary feature to ShareDB that manages to improve the accuracy metric substantially. In the next section we will discuss this feature.

8 Proprietary Solution

In light of the relatively weak performance in terms of accuracy by all the prior mentioned approaches, we proceeded to investigate improvements to ShareDB (OT) that could lead to reasonable accuracy. Eventually we determined that relaxing the consistency constraint leads to tremendous improvements in the result.

8.1 Periodic Consistency

One of the objectives of Real-Time collaborative editing as mentioned at the start of our report is eventual convergence. The goal is for the distributed states of the document to converge at some point in future. The exact nature of this convergence is not necessarily outlined in most texts related to real-time collaborative editing. This leads us to entertain the possibility of merging the states periodically in discrete time steps rather than attempt to continuously merge the state of distributed documents. We call this approach periodic consistency, in order to differentiate it from continuous consistency as demonstrated by previously investigated solutions. Periodic consistency has a few additional components on top of ShareDB's core components as outlined below.

8.1.1 Coordinator

The coordinator is a new service with the sole purpose to coordinate the periodic convergence procedure across all connected clients. For the purposes of our implementation we used the Presence API that is already part of ShareDB but the coordinator in theory can utilize any broadcasting technology that is suitable. The coordinator much like the server acts as a central service, but unlike the server only performs it's tasks periodically. At set intervals of t seconds the coordinator will perform the following tasks.

- Broadcast to all clients to pause execution of operations and provide the list of operations for relevant period.
- Clients will send the list of operations to the coordinator along with a time indicating their creation as per clients.
- The coordinator orders each of the operations by the creation time - start of client, this is equivalent to a total order algorithm.
- The coordinator then performs the operation on the document state at $t-1$ to arrive at document state of t.
- The coordinator then broadcasts this "converged" state for all clients to merge with their prior state.
- The coordinator broadcasts to all clients to resume executions of operations.

The above scheme allows for the convergence of state at set period intervals, this period can be modified by the user as per their need. The idea is to never let the state deviate too far from the intention, even if that entails periodic pauses to the ability to "edit" the document state by the clients. This does not dissipate however the "real-time" aspect as we still allow for OT to resolve the conflicts in real-time, our periodic merge is on top of the continuous merge that is provided by OT. This is to allow the clients to improve the accuracy of their collaborative editing periodically but still have a less than ideal accurate snapshot on a continuous basis. We will further discuss the advantages and disadvantages of our approach at a later stage.

8.1.2 Decentralized Clock

Perhaps the most important addition in our approach besides the coordinator is the concept of a decentralized clock. While the implementation of total order broadcast in Fluid did not yield positive results, we have utilized that idea within our own solution. We ensure that the final converged document has a total order on it's operation determined and dictated by the coordinator. This total order though is only determined at periodic intervals rather than continuously upon every operation as highlighted in the previous section. Each client maintains a local clock per document and upon subscription to the document sends the local time to the coordinator as per it's clock. The coordinator maintains this "time of subscription" for each client and utilizes this when ordering the operations. Each client utilizes their local clock to "attach" a time value to their operations to indicate their time of creation. At the end of the period as a client sends it's list of operations for that period to the coordinator, it includes this time of creation. The coordinator then simply orders the total set of operations for that period according to the formula below.

$$Order(OA, OB) = \begin{cases} OA, OB & \text{if } (OA.OT - OA.ST) < (OB.OT - OB.ST) \\ OB, OA & \text{otherwise} \end{cases} \tag{32}$$

Where OA and OB are two different operations and OT represents the client local time of the creation of that operation, while ST represents the client local time of subscription for the client of that operation.

This localized clock allows for better intention preservation as it does not suffer from degradation in quality due to network delay. Since the clock is localized the snapshot of intention is captured locally without relying on network communication. This does not however supersede the operational transformation used in ShareDB, as mentioned we still utilize OT for continuous integration of the document changes and hence the capture of the intention according to the clock is simply a complement to the state of the document at that point in time according to the client.

8.2 Advantages of Periodic Convergence

- The intention of the author of a particular operation is better preserved leading to better accuracy.
- The shared state of the document does not deviate from it's desired state for too long.
- Increased resilience against network delays and outages when compared to more traditional methods.

8.3 Disadvantages of Periodic Convergence

- The need to pause all clients for a short duration periodically, hence intrusive in nature.
- The interplay between intention resolution by both OT and TOB can get quite complex.

- The additional burden of maintaining a coordinator that is like the server a single point of failure.
- Increased utilization of both network and localized computation resources.

9 Accuracy Testing for Periodic Convergence

In this section we will outline the results of testing our solution similar to section 6, although in this section we will focus solely on the accuracy metric and provide comparative analysis to the base implementation of ShareDB.

9.1 Testing VM Specifications

The tests have been performed on a cloud based compute instance from Google Cloud Platform. The specifications are the same as in section 6. The specifications are as shown below:

- CPU: 2600 MHz Per Core (Quad Core)
- Memory: 16 GB

Benchmarks

In order to analyze and compare the performance of our solution relative to ShareDB, we have created separate applications to benchmark the basic implementation of ShareDB and our modified implementation. The basic implementation is the same one used in our previous tests in section 6.8. We further dockerize both these setups and execute them through docker compose in order to further analyze effect of various network disruptions. Each implementation uses the same insertion algorithm outlined in section 6.8 so as to keep the comparison fair. The various testing scenarios are listed below, for each scenario we repeat the test 10 times and take various statistical measures for purposes of comparison. The length of a period for our testing purposes is 60 seconds, hence to compare the basic implementation and our modified implementation 3 periods refer to 180 seconds of total insertions for both solutions. The network emulation is achieved using Pumba (https://github.com/alexei-led/pumba), which in turn utilized Netem and TC to emulate various network conditions in dockerized environments.

Results

This section outlines the results obtained through our testing, each section represents one of the below scenarios.

Scenario	Frequency	Users	Periods	Latency (ms)	Loss Rate (%)
A	2	25	1	0	0
B	2	50	1	0	0
C	1	50	1	0	0
D	2	25	3	0	0
E	2	25	2	500	0
F	2	25	2	0	10
G	2	25	2	500	10

9.1.1 Scenario A

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	7.57	2947	98.14	2843
2	16.11	2953	98.14	2842
3	12.52	2956	97.77	2865
4	12.11	2949	98.05	2873
5	22.33	2940	98.07	2844
6	12.12	2951	97.93	2855
7	4.15	2949	97.88	2834
8	8.19	2947	97.53	2838
9	12.25	2949	97.83	2851
10	21.89	2954	97.94	2866

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	12.924	12.185	21.89	4.15	5.86
Modified	97.928	97.935	98.14	97.53	0.19

9.1.2 Scenario B

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	4.04	5719	88.70	5081
2	6.04	5730	87.98	5041
3	4.14	5721	87.59	5028
4	0.31	5736	89.49	5090
5	2.20	5738	88.06	5093
6	4.11	5712	88.42	5103
7	0.26	5722	88.28	5045
8	4.09	5735	88.38	5042
9	4.01	5728	87.65	5143
10	4.12	5713	89.01	5051

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	3.33	4.07	6.04	0.26	1.84
Modified	88.36	88.33	89.49	87.59	0.59

9.1.3 Scenario C

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	10.19	2876	90.80	2566
2	8.20	2875	91.69	2600
3	6.28	2884	90.82	2561
4	4.32	2868	90.63	2560
5	12.15	2879	89.61	2541
6	8.26	2860	90.65	2578
7	2.39	2879	89.73	2572
8	10.30	2867	90.24	2557
9	6.42	2878	89.87	2549
10	8.22	2881	90.91	2552

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	7.67	8.21	2.39	12.15	2.92
Modified	90.50	90.64	89.61	91.69	0.64

9.1.4 Scenario D

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	12.11	8845	96.09	8786
2	20.07	8833	92.08	8787
3	4.22	8853	95.30	8786
4	8.12	8834	93.50	8764
5	8.13	8849	96.04	8772
6	12.12	8830	94.65	8768
7	16.08	8825	93.73	8776
8	4.18	8821	96.24	8754
9	8.10	8837	92.87	8782
10	8.22	8841	95.91	8778

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	10.135	8.18	20.07	4.18	5.04
Modified	94.64	94.98	96.24	92.08	1.51

9.1.5 Scenario E

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	0.82	5716	75.76	5791
2	15.82	5835	89.66	5853
3	12.37	5829	88.75	5671
4	11.09	5832	82.64	5690
5	11.26	5781	54.47	5713
6	14.18	5777	48.91	5682
7	12.45	5833	93.81	5833
8	1.96	5739	51.72	5714
9	0.26	5724	93.55	5828
10	12.93	5766	86.80	5749

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	9.31	11.82	15.82	0.26	5.90
Modified	76.607	84.72	93.81	48.91	18.01

9.1.6 Scenario F

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	12.48	5818	85.33	5813
2	8.31	5821	92.51	5781
3	15.30	5836	79.66	5791
4	8.15	5832	93.26	5799
5	12.71	5840	96.98	5799
6	13.41	5847	90.61	5827
7	8.62	5813	93.77	5833
8	15.80	5842	96.90	5809
9	8.05	5838	93.45	5829
10	10.06	5853	88.99	5779

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	11.29	11.27	15.8	8.05	3.02
Modified	91.15	92.89	96.98	79.66	5.34

9.1.7 Scenario G

Test Number	Accuracy (%) - Original	Doc Length - Original	Accuracy (%) - Modified	Doc Length - Modified
1	8.06	5858	95.72	5797
2	7.77	5830	92.51	5781
3	4.66	5821	93.00	5800
4	11.58	5793	88.90	5777
5	11.94	5820	77.66	5809
6	8.99	5849	85.02	5796
7	15.07	5825	81.54	5818
8	12.39	5842	96.68	5815
9	7.88	5827	85.38	5806
10	10.51	5835	88.34	5823

The below table further provides common statistical measures for the above dataset.

Library	Mean (%)	Median (%)	Max (%)	Min (%)	Standard Deviation (%)
Original	9.89	9.75	15.07	4.66	2.99
Modified	88.475	88.62	96.68	77.66	6.18

10 Conclusion

In our testing the addition of our coordinator has shown significant improvement in preservation of intention. This remains true even when we introduce artificial delays and network data loss using Pumba. We did observe a much higher degradation in performance when we introduced network delay but even in those circumstances, the maximum accuracy achieved by the original approach was lower than the minimum accuracy achieved by our modified approach. This does not however mean that our approach is better in all cases as there are many additional document editing scenarios which could be assessed. Testing all possible scenarios though is beyond the scope of this report and we defer additional testing to a later stage. Ofcourse the greatest weakness of our approach lies in the need to periodically pause the entire network from editing the document. This may not be feasible in some real world applications and hence they have to revert to the original ShareDB solution.

As our approach is a combination of total order broadcast and operational transformation it may improve further as both approaches become more mature. There are many steps that can be taken next, for instance optimizing the merge operation to limit the time required for the intermittent pauses of document editing or to making the total order broadcast mechanism more generic to allow for utilization of components other than a decentralized clock for intention calculation.

11 References

[1] Nicolaescu, Petru & Jahns, Kevin & Derntl, Michael & Klamma, Ralf. (2016). Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types. 39-49. 10.1145/2957276.2957310.

[2] Shapiro M., Preguiça N., Baquero C., Zawirski M. 2011 Conflict-Free Replicated Data Types. In: Defago X., Petit F., Villain V. (eds) Stabilization, Safety, and Security of Distributed Systems. SSS 2011. Lecture Notes in Computer Science, vol 6976. Springer, Berlin, Heidelberg.
`https://doi.org/10.1007/978-3-642-24550-3_29`

[3] Bartosz Sypytkowski. Delta-state CRDTs: indexed sequences with YATA.
`https://bartoszsypytkowski.com/yata/`

[4] Preston So. Yjs deep dive: What is Yjs and operational transformation? - Part 1.
`https://www.tag1consulting.com/blog/yjs-deep-dive-part-1`

[5] Preston So. Yjs deep dive: How Yjs makes real-time collaboration easier and more efficient - part 2.
`https://www.tag1consulting.com/blog/yjs-deep-dive-part-2`

[6] Preston So. Yjs deep dive: How Yjs handles offline editing and versioning - part 4.
`https://www.tag1consulting.com/blog/yjs-deep-dive-part-4`

[7] Kevin Jahns. Are CRDTs suitable for shared editing?
`https://blog.kevinjahns.de/are-crdts-suitable-for-shared-editing`

[8] C. A. Ellis and S. J. Gibbs. 1989. Concurrency control in groupware systems. SIGMOD Rec. 18, 2 (June 1989), 399–407. DOI.
`https://doi.org/10.1145/66926.66963`

[9] Srijan Agarwal. Building a real-time collaborative editor using Operational Transformation.
`https://srijancse.medium.com/how-real-time-collaborative-editing-work-operational-transformation-ac4902d75682`

[10] Srijan Agarwal. Operational Transformation, the real time collaborative editing algorithm.
`https://hackernoon.com/operational-transformation-the-real-time-collaborative-editing-algorithm-bf8756683f66`

[11] Fluid Framework, Total order broadcast & eventual consistency.
`https://fluidframework.com/docs/concepts/tob/`