VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

Aleksej Kazmin

BE KONFLIKTŲ REPLIKUOJAMOS DUOMENŲ STRUKTŪROS IR JŲ TAIKYMAS AUKŠTO PATIKIMUMO BENDRADARBIAVIMO PROGRAMINĖJE ĮRANGOJE

CONFLICT-FREE REPLICATED DATA TYPES AND THEIR APPLICATION IN THE DESIGN OF HIGHLY-AVAILABLE COLLABORATIVE SOFTWARE

Baigiamasis bakalauro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 612I13002

Informatikos studijų kryptis

Vilnius, 2019

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

TVIRTINU
Katedros vedėjas

____
(Parašas)

____
(Vardas, pavardė)

____
(Data)

Aleksej Kazmin

BE KONFLIKTŲ REPLIKUOJAMOS DUOMENŲ STRUKTŪROS IR JŲ TAIKYMAS AUKŠTO PATIKIMUMO BENDRADARBIAVIMO PROGRAMINĖJE ĮRANGOJE

CONFLICT-FREE REPLICATED DATA TYPES AND THEIR APPLICATION IN THE DESIGN OF HIGHLY-AVAILABLE COLLABORATIVE SOFTWARE

Baigiamasis bakalauro darbas

Inžinerinės informatikos studijų programa, valstybinis kodas 612I13002

Informatikos studijų kryptis

Vadovas prof. dr. Dalius Mažeika ______________ ______________
(Pedag. vardas, vardas, pavardė) (Parašas) (Data)

Vilnius, 2019

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ TECHNOLOGIJŲ KATEDRA

TVIRTINU
Katedros vedėjas

____
(Parašas)

____
(Vardas, pavardė)

____
(Data)

BAIGIAMOJO BAKALAURO DARBO (PROJEKTO)
UŽDUOTIS

………........................Nr. ....

Vilnius

Studentui (ei) ______________________
(Vardas, pavardė)

Baigiamojo darbo (projekto) tema: ______________________

patvirtinta 2019 m. kovo 6 d. dekano potvarkiu Nr. 145fm

Baigiamojo darbo (projekto) užbaigimo terminas 2019 m. birželio 3 d.

BAIGIAMOJO DARBO (PROJEKTO) UŽDUOTIS:

1. Investigate currently offered solutions to eventual consistency, design trade-offs, and implementation details.
2. Identify classes of applications for which CRDTs are beneficial.
3. Design, implement, and test reasonably practical peer-to-peer application using mathematically sound, compound CRDTs.
4. Compare costs and technical advantages of CRDTs over earlier approaches.

Baigiamojo bakalauro darbo (projekto) konsultantai: ______________________
(Pedag. vardas, vardas, pavardė)

Vadovas ______________________ (Parašas) prof. dr. Dalius Mažeika (Pedag. vardas, vardas, pavardė)

Užduotį gavau

____
(Parašas)

Aleksej Kazmin
(Vardas, pavardė)

2018 m. spalio 11 d.
(Data)

Vilniaus Gedimino technikos universitetas	ISBN ISSN
Fundamentinių mokslų fakultetas	Egz. sk. ....
Informacinių technologijų katedra	Data ..........-.....-.....

Pirmosios pakopos studijų **Inžinerinės informatikos** programos bakalauro baigiamasis darbas 3

Pavadinimas	**Be konfliktų replikuojamos duomenų struktūros ir jų taikymas aukšto patikimumo bendradarbiavimo programinėje įrangoje**
Autorius	**Aleksej Kazmin**
Vadovas	**Dalius Mažeika**

Kalba: anglų

Anotacija

Bendradarbiavimo programinė įranga leidžia nuotoliniams naudotojams realiuoju laiku dirbti grupėse bendro tikslo link. Tokios aplikacijos turi išlaikyti aukšto patikimumo lygį nepaisant tinklo nenuspėjamumų ir sutrikimų. Dažnai nusprendžiama sistemai apdoroti pakeitimus visais laiko momentais, bet leisti dokumentų būsenų nukrypimus tinklo skaidymo metu, automatiškai išsprendžiant juos kai ryšis yra atstatomas.

Yra sukurta daugybė sėkmingų bendradarbiavimo aplikacijų, pasitelkiančių centralizuota sujungimo schema. Tačiau, tokios aplikacijos linkę prastesniam patikimumui dėl aukštesnės rizikos taškų. Kita vertus, sudėtingi lygiarangio (angl. peer-to-peer) ryšio duomenų perdavimo būdai reikalauja formaliai verifikuojamų modelių. Tokių sistemų diegimo žinios iki šiol yra nepakankamos.

Šis baigiamasis darbas pateikia pagrindinius galiausiai neprieštaringų (angl. Eventually Consistent) sistemų iššūkius, palygina automatinių konfliktų sprendimo algoritmų OT (angl. Operational Transformation) ir CRDT (angl. Conflict-free Replicated Data Types) techninius privalumus bei kaštus. Tam, kad giliau ištirti CRDT panaudojimo atvejus, yra projektuojamas ir diegiamas lygiarangio ryšio teksto bendradarbiavimo redaktorius.

Sukurta aplikacija teisingai užfiksuoja naudotojo intenciją ir toleruoja ryšio nutraukimus. Testavimo rezultatai parodo, kad CRDT pasižymi puikiu laiko sudėtingumu ir yra pakankamai lengvai realizuojamos (apie 2000 TypeScript ir JavaScript kodo eilučių). Tačiau, jos atneša atminties naudojimo iššūkių. Taip pat, atminties naudojimo augimas nėra ribojamas dėl reikalavimo saugoti praeitų dokumentų būsenas. Tai gali būti išspręsta šiukšlių surinkimo (angl. Garbage Collection) algoritmais, tačiau dauguma sutiktų realizacijų pasirenka atsisakyti to dėl paprastumo.

Prasminiai žodžiai: Paskirstytasis skaičiavimas, CAP teorema, galiausiai neprieštaringos aukšto prieinamumo sistemos, OT, CRDT, RGA, bendradarbiavimo aplikacijos, priežastingumas, intencijos fiksavimas, lygiarangis ryšys.

Vilnius Gediminas Technical University
Faculty of Fundamental Sciences
Department of Information Technologies

ISBN ISSN
Copies No. ....
Date ..........-.....-.....

Bachelor Degree Studies **Engineering Informatics** study programme Bachelor Graduation Thesis 3

Title	**Conflict-Free Replicated Data Types and Their Application in the Design of Highly-Available Collaborative Software**
Author	**Aleksej Kazmin**
Academic supervisor	**Dalius Mažeika**

Thesis language: English

Annotation

Collaborative applications enable remote users to work in groups towards the common goal, in real time. Such applications have to remain highly available despite network delays or uncertainties. A frequent approach is to accept modifications at all times and allow state divergence during network partition, automatically resolving the conflicts when connection is restored.

Many successful collaborative applications were developed in a centralized topology. However, such systems are prone to outages due to presence of single points of failure. On the other hand, complex communication patterns of peer-to-peer topology require more formal, verifiable models. Knowledge of deploying similar applications is still scarce at best.

This thesis presents main challenges of eventual consistency and compares technical costs and advantages of conflict resolution under Operational Transformation (OT) and Conflict-free Replicated Data Types (CRDT). In order to investigate CRDT use cases in-depth, a peer-to-peer collaborative text editor is developed.

Implemented text editor correctly captures user intent and tolerates network partitions. Test results show that CRDTs have excellent time complexity and are relatively simple to implement (around 2000 lines of mixed TypeScript and JavaScript code). However, they bring their own challenges in the form of memory overhead. Furthermore, the memory growth is unbounded due to the requirement to store previous document states. This can be solved with garbage collection algorithms, although most encountered implementations choose to forego it for simplicity.

Keywords: Distributed systems, CAP theorem, eventual consistency, high availability, OT, CRDT, RGA, collaborative editing, causality, intent preservation, peer-to-peer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Modern Internet is an incredibly complex, interconnected world of its own. In the early days of the Internet the systems were simple - just a static webpage served to many consumers, which could be visualized as a one-to-many relationship. Web 1.0 had a shortcoming - the content was not dynamic. With adoption of Web 2.0, it became more common to provide means to website content modification, essentially transforming it into many-to-many relationship, where one group of users makes the changes, and other reads them.

Nowadays, software users expect to be able to work on the same content, at the same time, switching seamlessly between different mediums - be it a computer at home, smartphone, or an office workstation. Furthermore, humans inherently strive for collaboration. Connecting remotely distant locations often demonstrates physical limitations, imposing additional constraints on distributed system design. Such element of state replication and synchronization across devices extends many-to-many relationship to multiple dimensions. Consequently, it becomes impossibly hard to reason about all the different system interaction scenarios. Mathematically-sound structures provide the proof of correctness for algorithms used in distributed systems.

Connectivity is not always reliable - even the best hardware eventually fails, our network coverage is still not perfect, connection performance varies significantly from region to region. Therefore, while designing any kind of distributed system it is important to consider fault-tolerance, and, depending on use-cases, commit to certain trade-offs.

Earlier examples of web-based collaborative software, such as Google Docs and Apache Wave (originally Google Wave), rely on Operational Transformation (OT). OT is built on the premise that there exists a transformation function, which, applied to local and replicated operations, ensures convergence of all replicas towards a final state [1]. OT is a reasonable choice in a centralized topology, however, correctness of OT in distributed systems has been criticized numerous times, and it has been noted that "due to the need to consider complicated case coverage, formal proofs are very complicated and error-prone, even for OT algorithms that only treat two character wise primitives (insert and delete)" [2]. Correctness issues, along with the implementation difficulty in a truly distributed topology make OT not optimal for peer-to-peer software.

CAP theorem implies that under the constraints of availability and partition-tolerance (AP), distributed data store compromises on consistency (C) [3]. Conflict-Free Replicated Data Types (CRDTs) [4] are used to replicate data across many nodes, updating the state without the need for concurrency control. Any merge conflicts are guaranteed to resolve using a merge function which is associative, commutative, and idempotent. Thus, CRDTs provide Strong Eventual Consistency and monotonicity guarantees in such settings [4].

Goal

This thesis aims to analyze practical benefits of CRDTs and to propose improvements of the process of development of highly-available distributed collaborative applications.

Tasks

In order to reach the aforementioned goal, the thesis undertakes the following tasks:

1. Investigate currently offered solutions to eventual consistency, design trade-offs and implementation details
2. Identify classes of applications for which CRDTs are beneficial
3. Design, implement and test reasonably practical peer-to-peer application using mathematically sound, compound CRDTs
4. Compare costs and technical advantages of CRDTs over earlier approaches

1 STUDY OF HIGH AVAILABILITY IN COLLABORATIVE APPLICATIONS

1.1 Computer-supported cooperative work

Computer-supported cooperative work (CSCW) is a generic term, first coined in 1984, that "combines the understanding of the way people work in groups with the enabling technologies of computer networking and associated hardware, software, services and techniques." [6] [7] CSCW is an interdisciplinary field, which provides a framework for designing computer-based technology to support collaborative work.

Types of CSCW systems can be categorized based on a context of use cases. One such example is CSCW matrix, considering work contexts along two dimensions of time and space. Time dimension is either synchronous (everyone participates at the same time) or asynchronous (not all participants need to be present). Space is defined as co-located or geographically distributed. [8]

Table 1. Johansen's time-space CSCW matrix [8]

	Same time	**Different time**
Same place	Face to face interaction ● Digital whiteboard ● Wall display	Continuous task ● Team rooms ● Project management
Different place	Remote interaction ● Video conferencing ● Instant messaging chat rooms ● **Real-time groupware**	Communication & Coordination ● Version control ● Workflow ● Blog ● Email

This thesis focuses on the *same time / different place* interaction, specifically real-time groupware.

1.2 Industry movement towards high availability

Real-time groupware imposes certain constraints on the design of computer systems. From the usability point of view, users expect the software to reflect result of their actions immediately. For example, in video games, it is common to employ client-side prediction techniques in order to reduce effects of network latency. When processing user input, a client advances the game state locally, while waiting for the updated game state to return from authoritative server at the same time. If desynchronization would happen, local state is corrected using server state. [9]

Unfortunately, such approaches are not always suited for collaboration when a conflict arises due to concurrent operations. Consider a shared document. One actor deletes entire paragraph, while other appends a sentence to the end of it. There is not enough information to predict the outcome while preserving the meaning of a document. Some method of conflict resolution is required in order to agree on the final result.

1.3 CAP theorem

Existence of both pessimistic and optimistic approaches to data replication has long hinted at the fundamental trade-off between guarantees of distributed systems. Brewer's CAP theorem states that "in a shared-data system, it is only possible to provide two out of three guarantees": [3]

- Consistency (C) - Atomic consistency is a guarantee that "a total order must exist on all operations such that each operation looks as if it were completed at a single instance". [3] This implies that in case of a network partition, system must block until writes are propagated to all replicas before serving reads.
- Availability (A) - Availability is a guarantee such that "every request received by a non-failing node must result in a response". [3] In case of a network partition, a highly-available system is expected to continue processing requests, even if the data is stale.
- Partition tolerance (P) - implies that a system is able to continue processing data despite communication failures between replicas. Supporting offline editing in real-time groupware requires strongest partition tolerance guarantees.

Since network is subject to software and hardware failures, it is only possible to dismiss P under perfect conditions, therefore it is always a choice between A and C in practice.

It is important to note that a trade-off between A and C is not binary, but continuous. [12] For example, depending on use-case, a system might guarantee eventual consistency, or provide stronger read-your-writes consistency with monotonic reads.

More recent formulation extends CAP theorem to PACELC, stating that CAP is applied strictly in case of network partitions. However, partitions are "arguably rare", therefore another trade-off exists between latency (L) and consistency (C) when system operates normally. As soon as the system replicates data, it is subject to LC trade-off. [13] Consider multiple geographically distributed replicas. A write from one location might take hundreds of milliseconds to arrive to all replicas. As a result, to keep latency low, data is replicated asynchronously, which in turn sacrifices consistency.

1.4 Convergence, causality and intention preservation

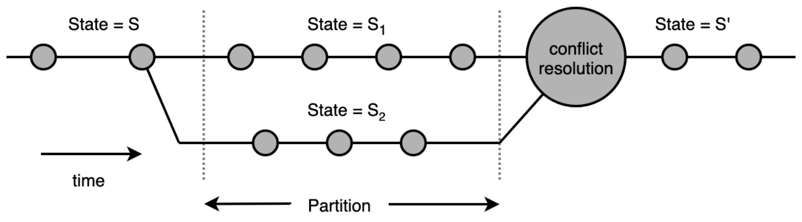


Figure 1. State divergence, conflict resolution, and convergence during network partition. [12]

As mentioned before, real-time groupware requires high availability, thus forfeiting consistency. Once eventually consistent system enters partition mode, actors begin to diverge. During the partition, two strategies are possible: “The first is to limit some operations, thereby reducing availability. The second is to record extra information about the operations that will be helpful during partition recovery.”. [12] It is not always possible to limit operations in order to preserve system invariants with local knowledge. To provide uninterrupted service, eventually consistent system accepts updates on all replicas during the partition. After communication is restored, the system converges towards the common state using some conflict resolution mechanism.

However, convergence by itself is not enough for intended effect to take place. For example, both actors could concurrently insert a word each, but the state would deterministically converge towards incomprehensible mix of characters based on causality information. Natural language carries large amounts of semantic context, which has to be captured in order to ensure intended effect.

Sun et al. (1998) define consistency model of a collaborative editing system as adherence to the following properties (CCI): [24] [25]

1. Convergence - “when the same set of operations have been executed at all sites, all copies of the shared document are identical”.
2. Causality preservation - for any pair of operations A and B, if $A \rightarrow B$, then A is executed before B on all sites.
3. Intention preservation - the effect of executing any operation A at all sites is the same as its original intention, and effect of executing A does not change effect of independent operations.

In practice, convergence and causality preservation can be achieved by ensuring total order and reliable causal broadcast at all sites. Such techniques are generic and well-understood. Achieving intention preservation, on the other hand, is much more complex and application-dependent. [24]

1.5 Conflict resolution under eventual consistency

Choice of conflict resolution method is entirely dependent on application requirements. The first intuition might be to attach a timestamp to each update. In case of conflict, an update with the greater time value is applied, while the other is discarded. This method is also known as Last-Write-Wins (LWW) and is widely used in eventually consistent databases such as Cassandra. However, as explained later, relying on wall clock has limited guarantees and may result in unintended data loss due to clock drift.

It is possible to delegate conflict resolution altogether by recording all conflicts instead of solving them. Distributed version control systems (VCS) such as Git favor it. In Git, non-concurrent changes to files are merged automatically, while conflicts are displayed to user. User then chooses the correct version manually. Conflict delegation is a reasonable choice if the resource contention is low, such as in VCS workflows, otherwise it may hinder application usability - consider two users editing same sentence in real time.

A natural improvement over LWW is to define precise merge semantics of all individual operations and all invariants. [12] This makes it possible to resolve conflicts automatically, without user intervention. Next section is going to take a deeper look into the most common automatic conflict resolution algorithms - Operational Transformation and Conflict-free Replicated Data Types.

1.6 Automatic conflict resolution

Google Docs is a well-known collaborative application. It allows users to concurrently work on the same part of a document in real time. It supports offline work, which is synchronized as soon as the connection is restored.

Google Docs collaboration functionality is largely based on Apache Wave (former Google Wave), which is an extension of Jupiter client-server collaboration system. [10] Its structure is described as “each wave is a collection of wavelets. A wavelet contains a collection of documents. A document consists of a structured, XML-like document and some annotations.”. Conflicts are resolved by applying Operational Transformation to wavelets. [10]

Wave’s operations describe changes to be performed on an empty document. An operation encapsulates a sequence of components, each applying a modification at the position of a cursor. Every operation component is relative to the cursor position, which enables streaming. [11] A subset of such components for text editing consists of:

- retain(positions) - advances the cursor a specified number of positions
- insert(characters) - inserts specified string at the current position, and advances the cursor to the end of inserted string
- delete(characters) - deletes specified string from the current index

Any two operations are composable iff they are compatible and satisfy the following identity:

$$(b \circ a)(d) = b(a(d)),\ \forall d \text{ on which } a \text{ can be applied} \tag{1}$$

where a, b - operations, d - document.

1.6.1 Operational Transformation

Operational Transformation (OT) is an optimistic concurrency control algorithm, which resolves conflicts without user intervention or locking. [11] It is defined as:

$$OT(c,\ s)\ =\ (c',\ s'),\ such\ that\ s' \circ c\ \equiv c' \circ s. \qquad (2)$$

where c - client operation, s - server operation, c'- operation dual to c, s' - operation dual to s.

By specification of OT, it is assumed that client and server can exchange operations sequentially at different times. Therefore, OT can be visualized using a diagram, in which client and server traverse through the state space via different paths, eventually arriving to the same convergent state. [10]

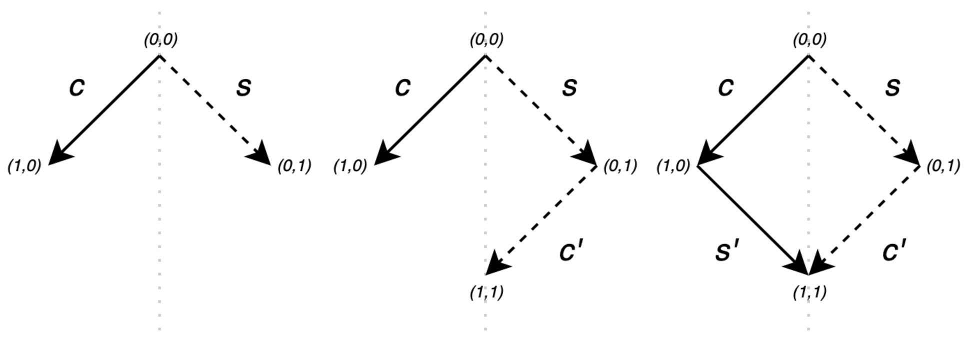


Figure 2. Client-Server state space. Time flows downwards, with client progressing towards the left, and server towards the right.

Initially, client and the server begin with synchronized state $(0, 0)$. As time progresses, client applies local operation c, while server receives an operation s from another client and applies it immediately. Both actors diverge and end up in states $(1, 0)$ and $(0, 1)$. Later, the server receives operation c. Server performs $OT(c,\ s)$ and obtains a pair of transformed operations $(c',\ s')$. Finally, it applies c' locally, and transmits s' to the client. Both actors converge to state $(1, 1)$.

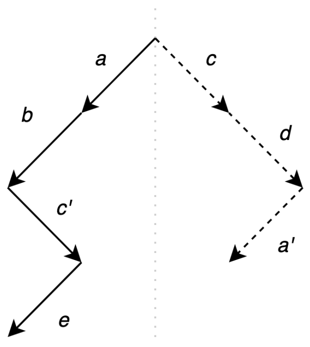


Figure 3. Multi-step divergence.

However, in real use cases, even in the presence of conflicts, many clients continue to perform and send concurrent operations to server. This results in multi-step divergences, which have to be resolved by transforming against phantom[1] operations and their compositions.

Putting OT to practice is difficult because there is a choice between covering all possible edge cases, which increases software complexity, or specializing it for concrete use-cases and dismissing formal proofs. [2] Lack of formal proof also imposes constraints on communication medium (such as requirement of a central server), which reduces their effectiveness and fault-tolerance in a truly distributed topology. It was also discovered that several OT algorithms do not satisfy stated convergence properties. [21]

1.6.2 Conflict-free Replicated Data Types

Conflict-free Replicated Data Type (CRDT) is a data structure which can be replicated across multiple nodes and does not require synchronization. CRDTs make use of mathematical properties that eliminate possibility of conflict, so that replicas provably converge towards a common state. Replicas remain available despite high network latency or faults. [4]

Many different simple CRDTs such as counters, sets, maps, and memory registers are formally defined. More complex CRDTs such as graphs and sequential buffers are also known to exist. [5] [14]

CRDTs are divided into two main categories:

- State-based Convergent Replicated Data Type (CvRDT)
- Operation-based Commutative Replicated Data Type (CmRDT)

[1] Phantom operations are understood as operations that have happened on another client, but which the server has not yet seen, so it must derive them optimistically in order to resolve conflicts.

CvRDT and CmRDT are equivalent - it is possible to emulate one through the other as a result of Strong Eventual Consistency (SEC). [4]

1.6.3 State-based CvRDT

By CvRDT theorem, “Assuming eventual delivery and termination, any state-based object that satisfies the monotonic semilattice property is SEC.”. [4]

Any structure can be considered monotonic semilattice iff it satisfies the following properties:

1. Set S of possible states is partially ordered by $\leq$ relation and there exists *least upper bound* (LUB) operation for all pairs in S
2. Merging local state s with remote state s' computes LUB of two states
3. State is monotonically non-decreasing across updates

These properties imply that a binary *merge* (also called *join*) operation $\vee$ provided by CvRDT is:

1. Associative: $x \vee (y \vee z) = (x \vee y) \vee z$
2. Commutative: $x \vee y = y \vee x$
3. Idempotent: $x \vee x = x$

CvRDT makes weak assumptions about the underlying messaging protocol. As a consequence, replicas can process messages in any order, and will converge towards LUB of most recent update. [4]

They are simpler to reason about, but require more bandwidth. This may not be optimal for large objects. Delta-CvRDTs can be used to only transmit differences, making them similar to operation-based approach. [14]

1.6.4 Operation-based CmRDT

By CmRDT theorem, “Assuming causal delivery of updates and method termination, any op-based object that satisfies the commutativity property for all concurrent updates, and whose delivery precondition is satisfied by causal delivery, is SEC.”. [4]

CmRDTs replicate state by propagating operations to other replicas. Instead of *merge* function, a pair of *generator* and *effector* functions are used. When a source replica witnesses an event, it executes *generator* function to obtain *effector* - an encoded side-effect of the operation. The *effector* must then be executed on all replicas. [14]

Replicas are guaranteed to converge if *effector* is delivered in causal order provided concurrent operations commute. Causal delivery can be forfeited if all operations are commutative. [4]

Compared to CvRDTs, CmRDTs use less bandwidth and may provide superior performance. Their implementation is also simpler. However, they require reliable causally-ordered broadcast communication protocol from messaging middleware. [4]

Some data types, such as Counters, commute naturally. However, because operations are often non-commutative, CRDTs must define precise concurrency semantics. [14] For example, for a Set data type, it is possible to have several convergent states resulting from concurrent *add* and *remove* operations. Applications must decide on the logical order relation for operations. Possible semantics for the Set are *add-wins*, *remove-wins* or *last-write-wins*. [14]

Last-write-wins semantics are available if operations follow total order. Generally speaking, total message ordering between computers in a distributed system is equivalent to consensus. However, total order can be approximated by ensuring *happened-before* relation (partial order), and employing additional ordering by unique identifiers or wall-clocks when messages are concurrent. [15]

1.6.5 Practical usefulness of CRDTs

Generally speaking, CvRDTs are often used in key-value stores such as Redis, Dynamo and Riak and file systems such as NFS, AFS, Coda, IPFS, while CmRDTs are more suited towards collaborative software. [4]

Counters are popular for counting the number of currently logged in users, activity tracking. They can also be used to implement highly available voting systems.

Append-only sets are used in blockchain technologies. Two-phase sets can model shopping cart in a collaborative grocery application, with one set for adding items, and another for removing. In case of concurrency “remove wins” semantics can be used.

Registers are used for distributed caching, in which frequently accessed resources get replicated to every region.

Sequence CRDTs such as TreeDoc, Replicated Growable Array (RGA), WOOT, Logoot, LSEQ were designed to be used in collaborative text editors, as they allow to preserve total ordering between text elements.

1.7 Event causality in distributed systems

Computers in a distributed system are subject to failures. Network cannot be assumed to be reliable, homogeneous or exhibit zero latency at all times. [16] Two messages A and B sent between computers might arrive in any possible order. Thus, it is required to include certain metadata, which would allow to process messages in the correct order.

One common approach is to attach wall clock timestamps to totally order messages. However, wall clocks are not consistent across the network. Furthermore, wall clocks occasionally violate monotonicity and may flow backwards due to leap second corrections. To mitigate this, Unix-based system clock can be configured to be monotonic, such that backward jumps are not observed in case of correction. [18]

Usually, computers track time using crystal oscillators. All crystals show minor differences in their oscillation frequency. To solve this issue, Network Time Protocol (NTP) is used for time synchronization against central reference clock. However, NTP only reduces, but does not completely eliminate time drift. Latest NTP iteration, NTPv4 promises accuracy of up to the "tens of microseconds with modern workstations and fast LANs". [17] When considering geographically distributed networks, there are fundamental limitations to how fast signals can travel, reaching remote locations in order of hundred milliseconds, which places an upper-bound on accuracy of time synchronization.

1.7.1 Logical clocks

Logical clocks are a suggested alternative to wall clocks. They do not measure real time, but instead capture causality between events of a system.

Happened-before relation between events in different processes can be tracked with the help of logical clocks. The *happened-before* relation $A \rightarrow B$ of event A *causally affecting* event B is defined as: [15]

1. If A and B are sequential (within same process) and A comes before B, then $A \rightarrow B$
2. If A is the sending of a message by one process, and B is the receipt of that same message by another process, then $A \rightarrow B$
3. If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$

Finally, if $\overline{A \rightarrow B}$ and $\overline{B \rightarrow A}$, then events are said to be *concurrent* ($A || B$). [15] Lamport clocks achieve this by maintaining incrementing counter on each process. Suppose C_i is the Lamport clock, initially 0, for process P_i. The following rules are used to implement Lamport clock: [15]

- If $A \rightarrow B$ in P_i, then $C_i(B) = C_i(A) + 1$
- If A is event of sending message m from P_i, then a timestamp is attached $t_m = C_i(A)$
- If B is event of receiving message m in P_j, then $C_j(B) = max(C_j(B),\ t_m + 1)$

An important shortcoming of Lamport clocks is that while $A \rightarrow B$ implies $C(A) < C(B)$, $C(A) < C(B)$ **does not** necessarily imply $A \rightarrow B$. Vector clocks generalize Lamport clocks into the context of multiple parallel and independent processes and solve this problem.

Let N be the number of processes. Each process P_i maintains vector clock V_i of length N. $V_i[i]$ is P_i clock value, while $V_i[j]$ $(i \neq j)$ is P_i approximation of logical clock for P_j. Vector clock implementation rules closely resemble those of Lamport's clock: [19] [20]

- If $A \rightarrow B$ in P_i, then $V_i[i](B) = V_i[i](A) + 1$
- If A is an event of sending message m from P_i, then a vector is attached $T_m = V_i(A)$
- If B is an event of receiving message m in P_j, then $V_j[k](B) = max(V_j[k](B),\ T_m[k]),\ \forall k$

1.7.2 Causal ordering of messages

Birman-Schiper-Stephenson (BSS) protocol defines algorithm for reliable causally-ordered broadcast communication and builds upon the idea of vector clocks. [22]

Before broadcasting message m, process P_i increments $V_i[i]$ and timestamps m with $T_m = V_i$. When P_j, where $j \neq i$ receives m, it delays its delivery until both conditions are true:

1. $V_j[i] = T_m[i] - 1$
2. $V_j[k] \geq T_m[k],\ \forall k \neq i$

Queued messages are sorted by vector clock, with arbitrary globally unique property used to break the ties for concurrent messages. When m is delivered at P_j, V_j is updated according to vector clock merge operation.

1.8 Conclusions

A framework of different modes of computer-supported cooperative work was presented. To support low latency real-time collaboration, applications tend to favor availability over consistency (as per CAP theorem). This places tight constraints on the design of core data structures such that correctness is not compromised. One of the main challenges of this field is choosing appropriate conflict resolution strategy. Two of the most recently studied approaches to automatic conflict resolution - Operational Transformation and Conflict-Free Replicated Data Types were investigated.

Both algorithms solve same challenge through different means - OT deals with the problem of conflicting edits in increased time complexity, while CRDTs carry additional metadata in order to resolve conflicts, which increases their space complexity. In general, it is not possible to clean-up stale metadata (e.g. tombstones for deleted items) while processing messages at the same time. Therefore, most implementations may need to employ garbage collection algorithms with some form of consensus or locking. Fortunately, CRDTs allow to perform this step asynchronously.

Research on CRDTs is still a relatively new area. In certain cases practical applications of CRDTs are superior to OT because they have clearly defined properties and are simpler to implement and prove. However, they tend to have higher memory usage, and not every use-case can be modelled using CRDTs.

2 DESIGN AND IMPLEMENTATION

2.1 System requirements

2.1.1 Use cases

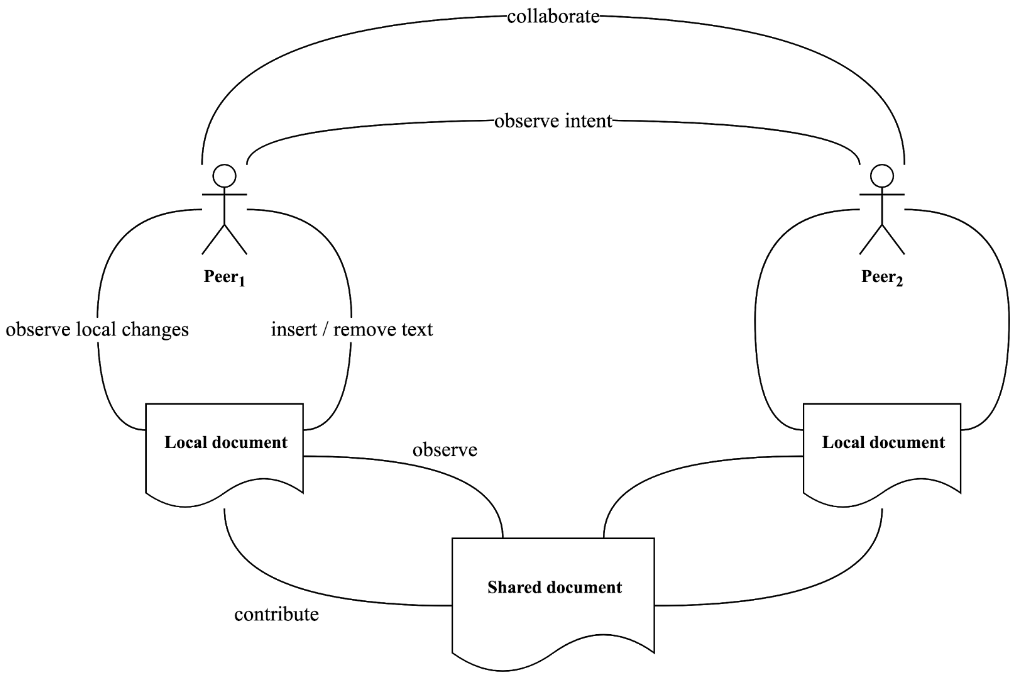


Figure 4. Collaborative text editor use cases. Because all operations are performed on local documents, shared document is only conceptual and does not exist as a materialized entity.

Table 2. Use cases of a collaborative text editor. Remote use cases happen asynchronously over network.

Use case No.	Use case	Local	Remote
U1	Insert text	+	
U2	Remove text	+	
U3	Observe local changes	+	
U4	Contribute to shared document		+
U5	Observe shared document		+
U6	Collaborate with others		+
U7	Observe intent of others		+

2.1.2 Functional requirements

Table 3. Summary of functional requirements.

UC No.	Req. No.	Requirement	Input	Result	Persisted data
U1	F1	Text can be inserted at any position of a document.	Sequence of characters	Characters inserted in CRDT	Characters with corresponding logical timestamps
U2	F2	Any part of a document can be selected and removed.	Selection range	Removed characters marked as hidden	Hidden flag is set on removed characters
U3	F3	Local changes are observed immediately, at all times.	Local operations	Text sequence is reconstructed from CRDT	None
U4	F4	Local changes are transmitted to other peers immediately.	Local operations	Local operations are sent to other peers	Vector clock
U5	F5	Changes of other collaborators are automatically integrated into local document as they are received.	Remote operations	Characters inserted or removed in CRDT	Characters with logical timestamps
U6	F6	In absence of new changes, all collaborators eventually observe the same document state.	Operation on a single replica	Operation broadcasted to all replicas	Same elements on all replicas
	F7	Operations are delivered in causal order.	Operations received out-of-order	Causal order is restored using vector clock	Vector clock
	F8	Original intent is preserved globally.	Concurrently edited text	Concurrent characters are grouped by peer	Set of observed logical timestamps
U7	F9	Document caret or selection of collaborators is shown in real-time.	Caret start and end positions	Positions stored in replicated LWW-Register CRDT	Caret positions with corresponding timestamps

In order to investigate practical application of CRDTs, a collaborative text document editor is developed. Formally, a collaborative editor provides a shared document (Figure 4). Document is defined as an ordered set of elements D, on which each user can perform the following operations: [23]

- $insert(e,\ i)$ for $e \in D$ and $i \in \mathbb{N}$: inserts element e at position i
- $remove(i)$ for $i \in \mathbb{N}$: removes element at position i
- $read$: retrieves contents of a document

Text editor tracks the position of each operation through the text caret. However, insertion position does not provide total order because it is local to each user. In other words, the position of the same text element is eventually consistent across all replicas. To ensure **convergence**, underlying data structure has to define unique, totally-ordered identifiers for each operation. As a consequence, the presentation layer needs to transform local caret position into unique identifier.

Because of high-availability requirement, the system has to correctly operate under the model of eventual consistency. The underlying transport-layer is assumed to be best-effort, meaning that it does not provide any delivery or ordering guarantees. To ensure **causality**, application messaging middleware must implement reliable causal broadcast.

Intention of text modifications must be preserved at all sites through **intent preservation**, such that locally-observed effect can be observed at any other replica.

Real-time caret position is a less critical component of the system. It is provided to improve collaboration between peers. Therefore, it is implemented using a less robust LWW-Register CRDT based on wall clock timestamps.

2.1.3 Non-functional requirements

Table 4. Summary of non-functional requirements.

Use case	Assoc. functional requirements	Non-functional requirements
Performance	F1, F2, F3	● Local changes are reflected immediately, as if they were written in an offline editor. ● Local input latency should depend solely on local hardware, not external factors such as network. [26]
Working offline	F1, F2, F3, F6, F7, F8	● Local document can be edited offline for indefinite period of time. ● Changes are synchronized with other users when connection is restored.
Privacy	F4	Text document is only shared with collaborators - it is not stored in any cloud.
Fault-tolerance	F6	Text data survives if at least one collaborating user has a copy of it.
Usability	F5	Changes of other users do not interrupt local typing flow.

Collaborative editor has to be highly-available, meaning that it is able to process user input immediately, even when offline. Changes from other peers are asynchronously integrated at some later point in time.

In order to improve performance, privacy, and fault-tolerance, application will send and receive data in peer-to-peer fashion.

Better performance is achieved through a more flexible network path. Because there is no central server, network packets can travel through the most optimized path between the peers. Furthermore, if a group of peers is located in the same local area network - there is no need to cross network boundaries.

Peer-to-peer communication also enables privacy, understood as the total ownership of the personal data. The editor data is shared only with collaborating parties.

Fault-tolerance is provided by redundancy in the system. In a fully connected network, all nodes are interconnected and form a complete graph. Each peer contains full history of document changes, which can be shared with new participants. As a result, the system is able to tolerate up to $(n-1)$ lost nodes. However, it has the greatest cost, as the number of connections grows quadratically: $c = \frac{n(n-1)}{2}$.

Edited document is expected to consist of mostly textual data with limited number of participants, thus bandwidth constraints can be relaxed in favor of more robust full-mesh network topology.

2.2 System architecture

2.2.1 Actors

Application is contained entirely in a web browser, so that it can be accessed through any modern device such as workstation, laptop, or a smartphone.

At first, users connect to a static file server, which distributes peer-to-peer application and other static assets through HTTP protocol. Then, application establishes a WebSocket connection to the central signaling server. Signaling server performs automatic service discovery for instances of application, allowing them to initiate peer-to-peer connection through WebRTC.

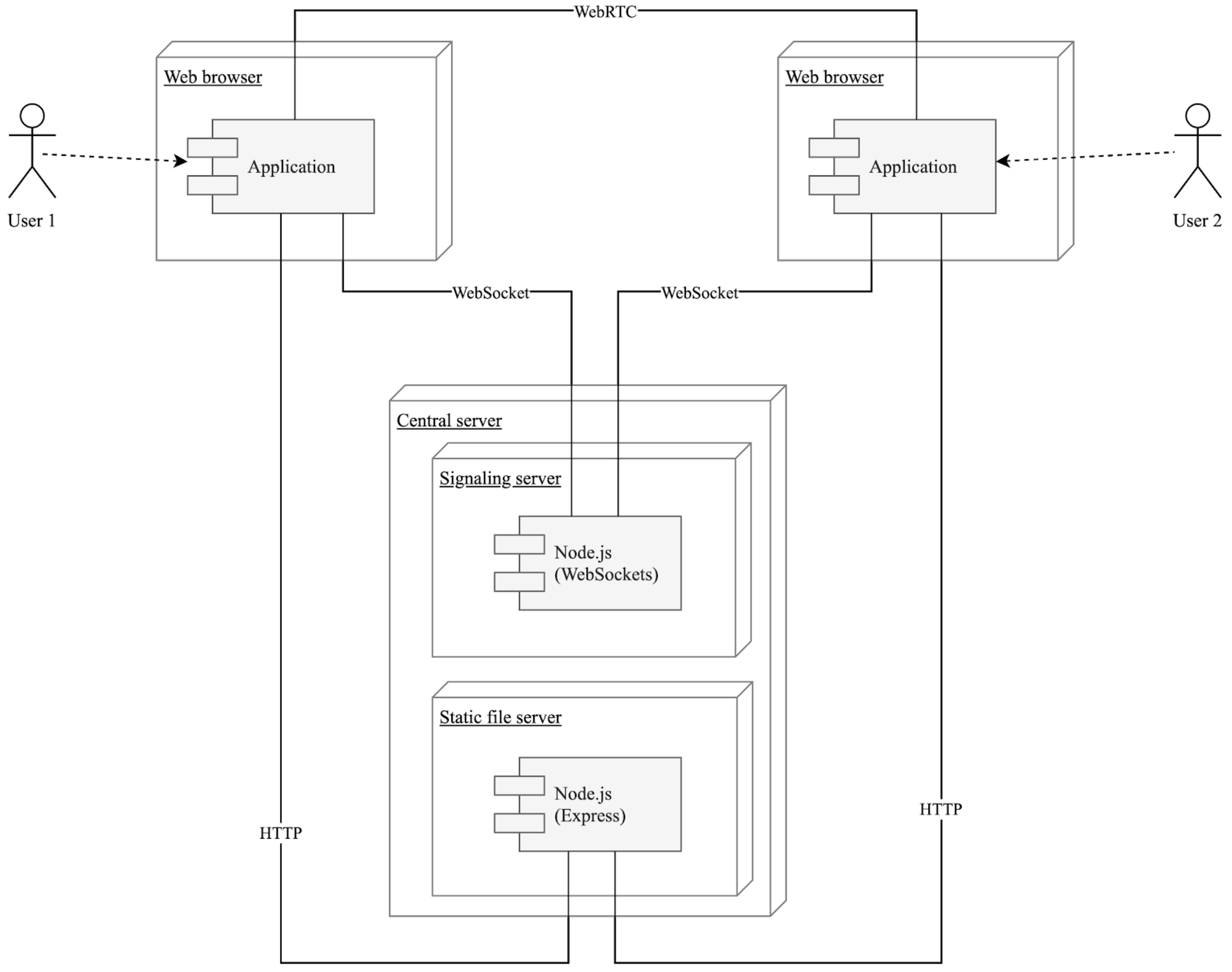


Figure 5. Actors of a system and their communication protocols.

2.2.2 Technologies

The peer application will be developed using JavaScript. To keep technological stack simpler, signaling server will also be developed using JavaScript and run in Node.js environment. The choice of technologies is motivated by the excellent portability and developer support of modern browsers, as well as mature and standardized technologies:

1. WebSockets - for bidirectional stateful communication with signaling server.
2. WebRTC - for peer-to-peer communication between peers.
3. React - for reactive component-based interface.
4. Express.js - for serving peer application and static assets.

Since TypeScript offers static typing and compiles to JavaScript, it will be applied in critical areas to reduce risk of errors. Additionally, space-efficient Cap'n Proto[2] binary data interchange format will be used between peers, because text formats such as JSON or XML are expensive for frequent operation-based communication.

2.2.3 Peer topology

Each peer maintains a WebRTC connection to every other peer. Initial connection and session metadata exchange is established using a WebSocket connection to a signaling server (Figure 6). When the peers are fully connected through the WebRTC protocol, they can begin exchanging data, and signaling server is no longer needed.

Data is exchanged in both directions through WebRTC DataChannel by broadcasting it to every connected peer.

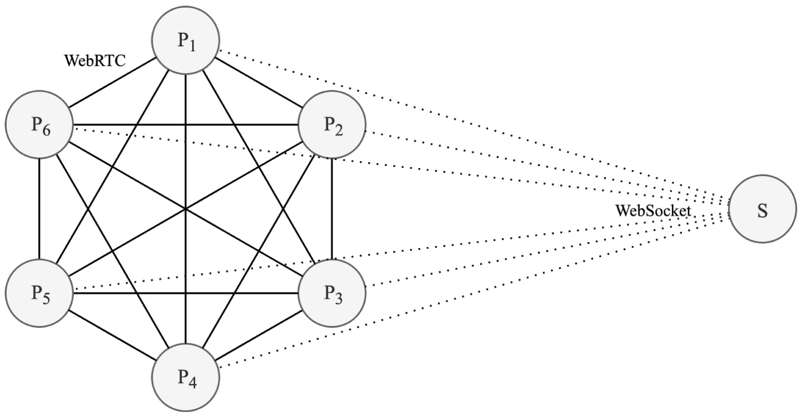


Figure 6. Full-mesh network.

[2] https://capnproto.org/

2.2.4 Signaling scheme

Central signaling server fulfills the role of a stateful router. It manages a list of currently connected peers and delivers handshaking messages between them. The process of connecting any two peers together can be summarized in 3 steps: [30]

1. Peer discovery
2. Offer
3. Answer

A connecting peer initiates connection by first announcing itself to a network. Server begins tracking it and replies with a list of other peers. Then, initiator sends a connection offer to every peer in a list. A WebRTC connection is established when another peer accepts the offer.

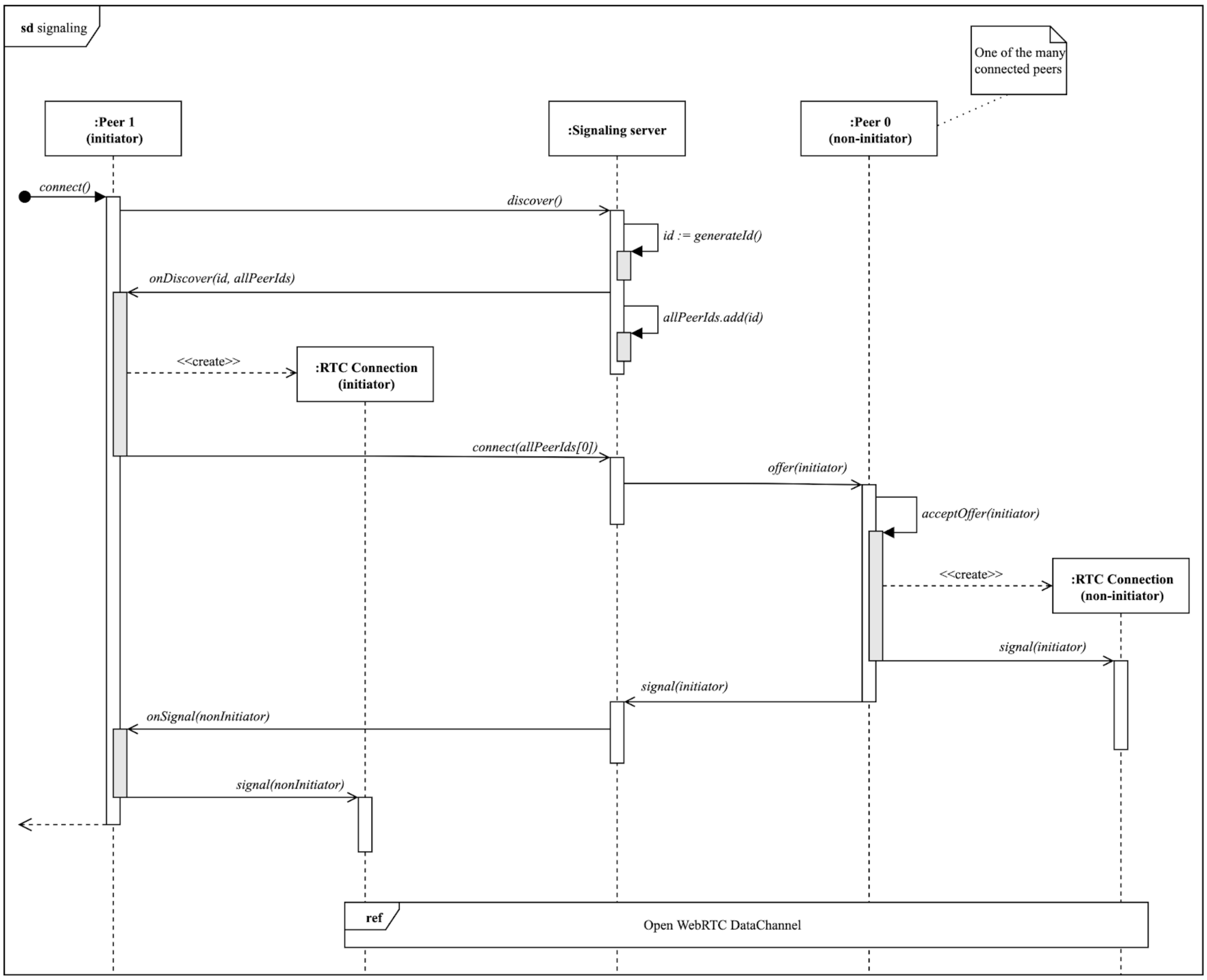


Figure 7. Signaling sequence for a connecting peer.

Once peers are connected directly, data channel can be opened for bidirectional data transmission.

2.2.5 Peer architecture

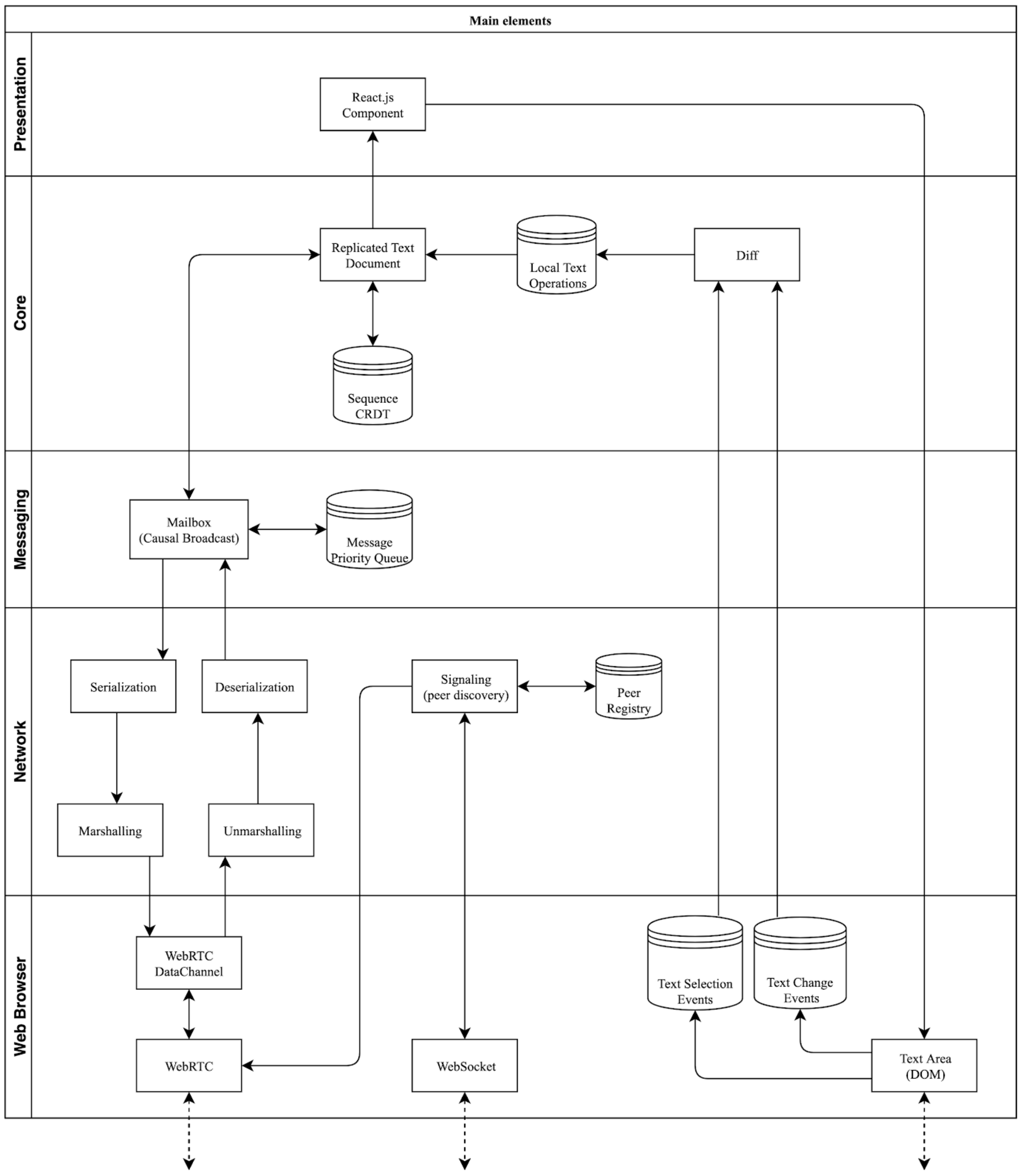


Figure 8. Peer architectural layers. Web Browser is included for clarity.

The architecture of peer application (Figure 8) is multi-layered. Each layer is separated by concerns:

1. **Presentation** - composes view of a text document and renders it to HTML DOM.
2. **Core** - maintains state of text document in a sequence CRDT. Transforms HTML DOM events to internal index-based representation.
3. **Messaging middleware** - responsible for message delivery. Ensures that received messages are processed in causal order. Enriches sent messages with causal metadata.
4. **Network** - responsible for data transmission between peers. Handles low-level concerns such as serialization, marshalling, and connection establishment.

The system can be viewed as local and remote feedback loops, each operating asynchronously through different channels. As local text is edited, it is immediately integrated into CRDT, while being asynchronously broadcasted to other peers. When remote operations are received, they are integrated into CRDT without disrupting local experience.

2.2.6 Initial synchronization scheme

To make any meaningful contribution, a new peer has to observe previous document state. During signaling, full history is requested from existing random peer (Figure 9). Existing peer deconstructs full CRDT state into individual operations, such that it is possible to reconstruct it on the receiving end. Because of data channel limitations on packet size, it has to be streamed in small chunks.

Here, causal delivery of chunks between two peers must be guaranteed as part of the messaging middleware.

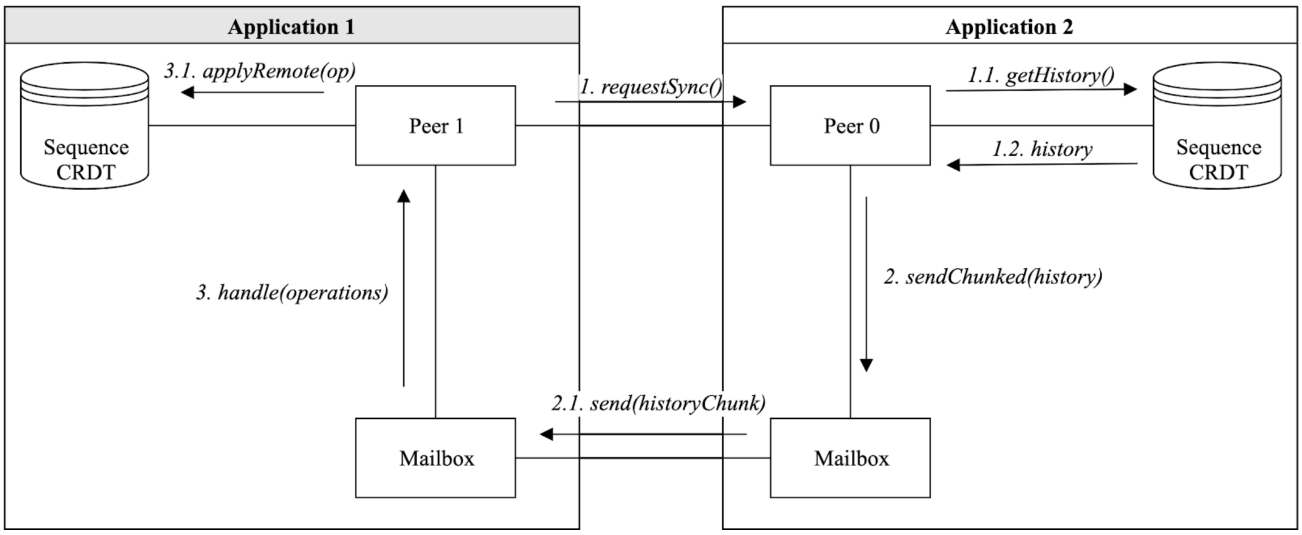


Figure 9. Full history synchronization scheme for a recently joined peer.

2.2.7 Sequence CRDT

As mentioned before, a text document is an ordered set of elements, each representing a character. Because of possibility for concurrent operations, either because of network latency, or concurrent editing behaviour, the CRDT has to preserve **total order** between inserts. Total order between operations guarantees that all replicas observing the same operations will converge to the same state.

Replicated Growable Array (RGA) is a linked-list like structure, in which each node contains an element, identifier, and identifier of the following node. [29] In order to support delete operations, each node additionally contains a tombstone flag.

Identifier t is a pair of unique site (peer) id and a sum of vector clock value during insertion. [29] When inserting new element, a reference identifier is first located. Then, a node with the new identifier is inserted to the right of it. Finally, an insert operation is serialized and transmitted to other peers.

Total order between identifiers is defined as: [29]

$$(site,\ clock) < (site', clock') \text{ if } (clock < clock') \tag{3}$$

Otherwise, if clocks are equal, site id is used for comparison:

$$(site,\ clock) < (site', clock') \text{ if } (clock = clock') \wedge (site < site') \tag{4}$$

RGA requires certain optimizations to be useful in practice. [29] A similar, more general data structure, was independently described by Attiya et al. (2016) as Timestamped Insertion Trees [27] and as Causal Trees by Grishchenko [31].

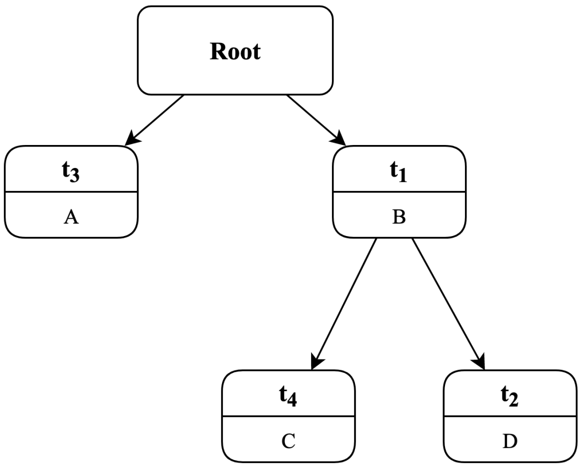
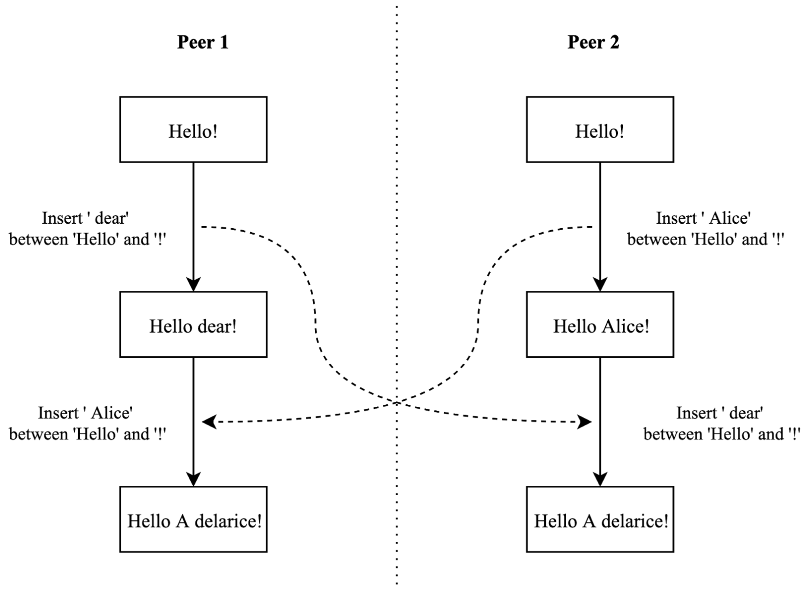


Figure 10. Timestamped Insertion Tree storing a text sequence *ABCD*. Order of identifiers is $t_1 < t_2 < t_3 < t_4$. Reading stored text involves traversing tree depth-first (pre-order) and visiting children in order of descending identifiers. [27]

The semantics and identifier structure is the same as in RGA, but the structure is easier to reason about. The tree itself need not be stored in memory as recursive structure - contiguous arrays may be used (trading insert for read performance). It also replaces vector clocks in identifiers with Lamport clocks because of significantly lower overhead.

It is important to note that tree structures assume causal delivery guarantees, because a children node might get inserted before parent is created. Causal broadcast implementation is described in the next chapter.



Peer 1
Peer 2
Hello!
Hello!
Insert ' dear' between 'Hello' and '!'
Insert ' Alice' between 'Hello' and '!'
Hello dear!
Hello Alice!
Insert ' Alice' between 'Hello' and '!'
Insert ' dear' between 'Hello' and '!'
Hello A delarice!
Hello A delarice!

Figure 11. Per-character interleaving anomaly during concurrent editing. Dashed lines represent merging process of remote operations. [28]

Unfortunately, it was later discovered that many sequence CRDT specifications, including RGA, suffer from interleaving anomalies during concurrent editing. [28] Figure 11 shows concurrent editing scenario in which user intent is not preserved, despite replicas correctly converging to the same state.

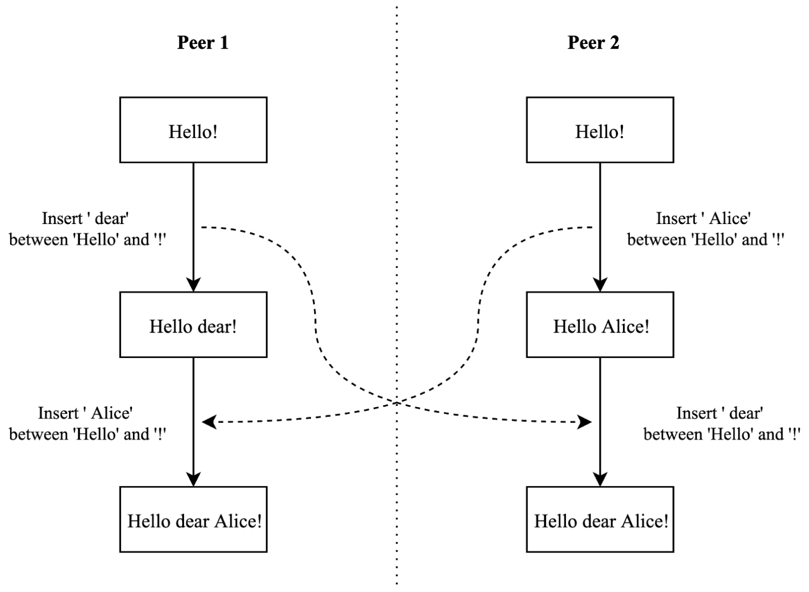

Peer 1
Peer 2
Hello!
Hello!
Insert ' dear' between 'Hello' and '!'
Insert ' Alice' between 'Hello' and '!'
Hello dear!
Hello Alice!
Insert ' Alice' between 'Hello' and '!'
Insert ' dear' between 'Hello' and '!'
Hello dear Alice!
Hello dear Alice!

Figure 12. User intent is preserved by grouping concurrent inserts by source replica. [28]

To remedy this problem, an additional set of identifiers e, containing observed children of reference node at the moment of insertion (in essence, capturing happened-before relation), is attached to each node. [28] Total order relation is also reformulated: [28]

$$(t_1,\ e_1) < (t_2,\ e_2) \text{ if } t_1 \in e_2 \tag{5}$$

$$(t_2,\ e_2) < (t_1,\ e_1) \text{ if } t_2 \in e_1 \tag{6}$$

Otherwise, if inserts are concurrent:

$$m_1 = min(\{t_1\} \cup e_1 - e_2) \text{ and } m_2 = min(\{t_2\} \cup e_2 - e_1) \tag{7}$$

$$(t_1,\ e_1) < (t_2,\ e_2) \text{ if } m_1 < m_2 \tag{8}$$

$$(t_2,\ e_2) < (t_1,\ e_1) \text{ if } m_2 < m_1 \tag{9}$$

As a result of this change, concurrent operations are grouped by individual peer and cannot be interleaved on a per-character basis (Figure 12).

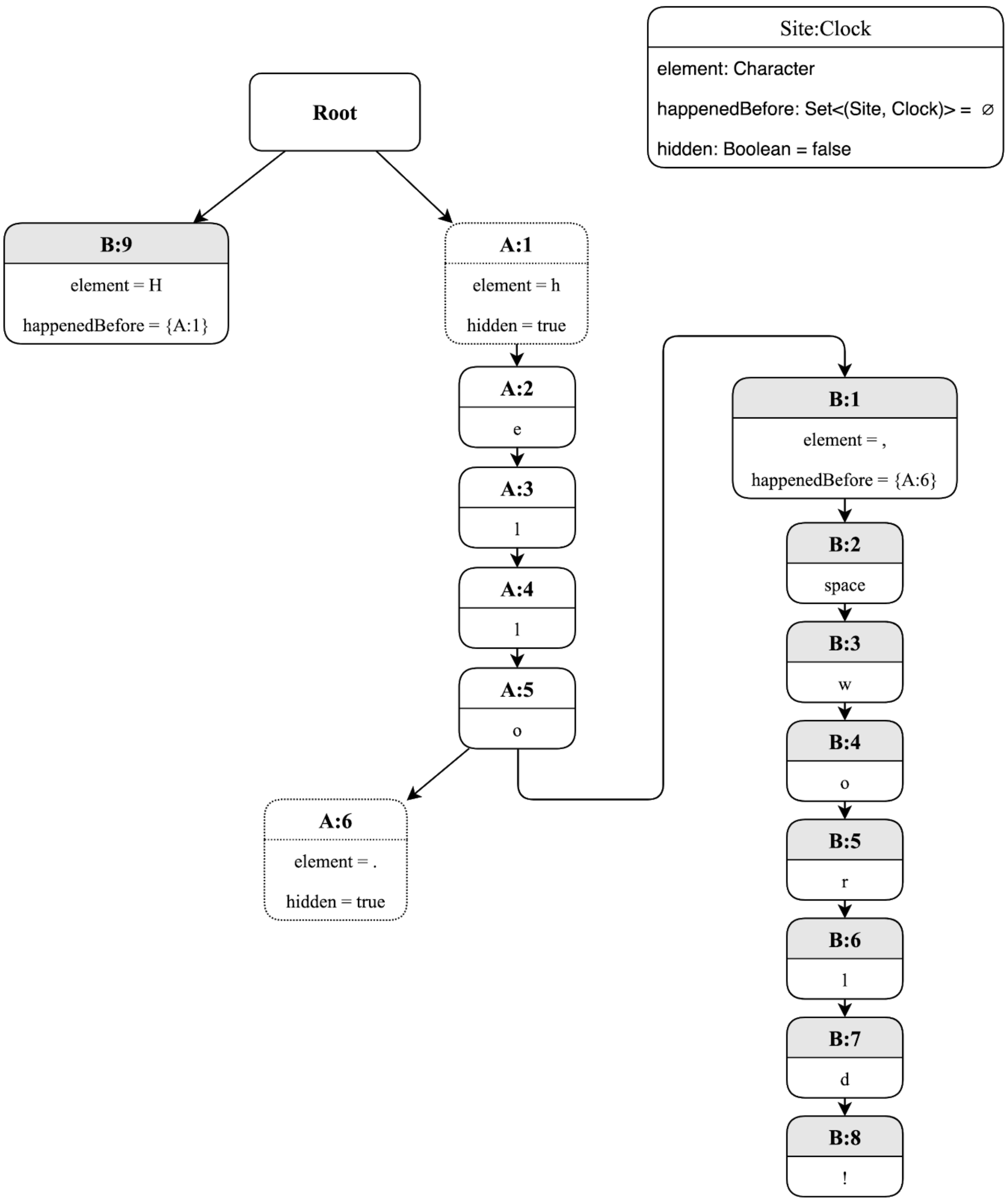


Figure 13. Internal causal tree representation of typed text. Default values in nodes omitted for brevity.

Figure 13 illustrates a text sequence being edited by two peers A and B. Peer A types text ‘hello.’. Later, peer B replaces ‘.’ with ‘, world!’ and capitalizes the first character resulting in a string ‘Hello, world!’. Removed characters are marked as tombstones instead of being deleted.

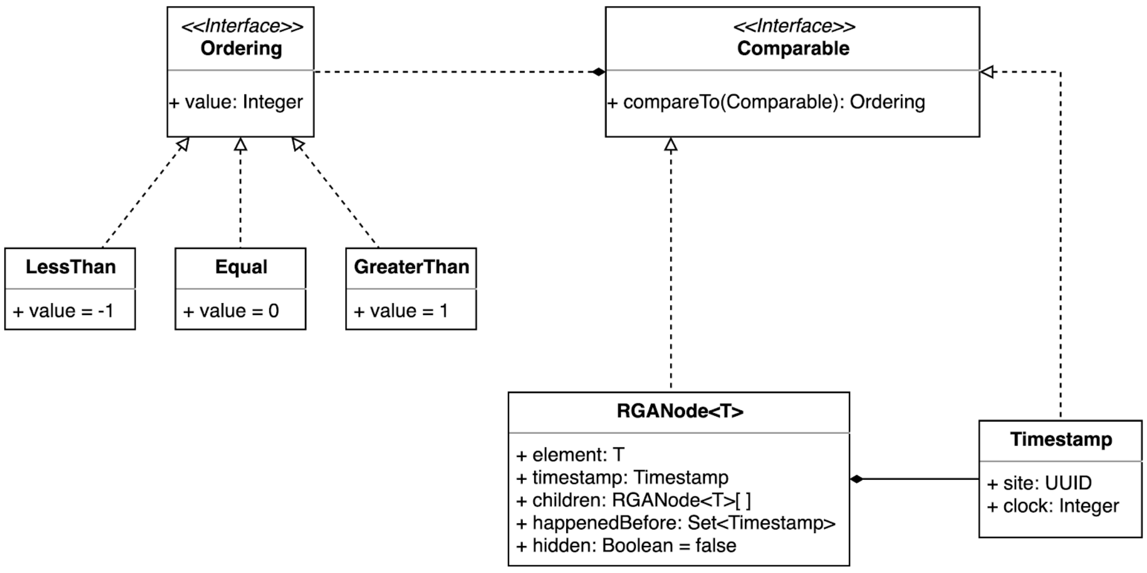


Figure 14. Definition of a tree node. Nodes are comparable and form total order based on a set of seen operations and its logical timestamp.

In current implementation, such tree is represented as a recursive data structure, where each node contains a pointer to its children. A structure of a single tree node is presented in Figure 14. Although it may not be very efficient in terms of CPU cache and memory usage, the resulting code is much simpler.

2.2.8 Transformation of local text changes to local positional operations

Core application layer communicates with presentation layer through a position-based document API. Because browsers do not provide such functionality for Text Area DOM elements, a transformation function is needed to bridge these interfaces together.

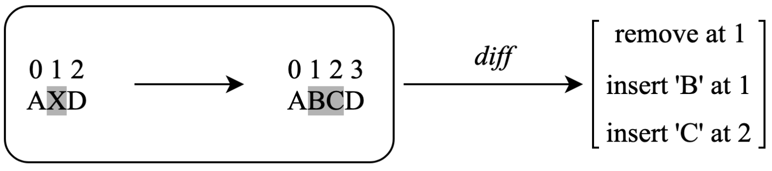


Figure 15. Changed text to operations transformation function. For illustration purposes each character is annotated with its position, with modified characters being highlighted.

The transformation function receives previous and the current version of a text fragment along with caret position and produces a sequence of atomic positional operations. These operations can then be processed by the core layer.

2.2.9 Lookup of global CRDT identifier by local text position

In order to insert a character at some position in a document, it is necessary to first locate the corresponding node in a tree. The naive approach is to traverse the tree depth-first, counting visible elements, until the position is found. However, such solution has $O(N)$ complexity, with N being the number of inserted characters (including tombstones). [29] As this operation is performed fairly frequently, it may quickly become a bottleneck in larger documents.

A suggested optimization is to additionally keep a weighted binary search tree, with weight being a total size of the subtree. [29]

2.2.10 Integration of remote operations

Since timestamp identifiers of tree nodes are unique and form total order, a mapping of $Timestamp \mapsto Node$ is maintained in a hash table alongside tree to support quick node lookup by timestamp identifier.

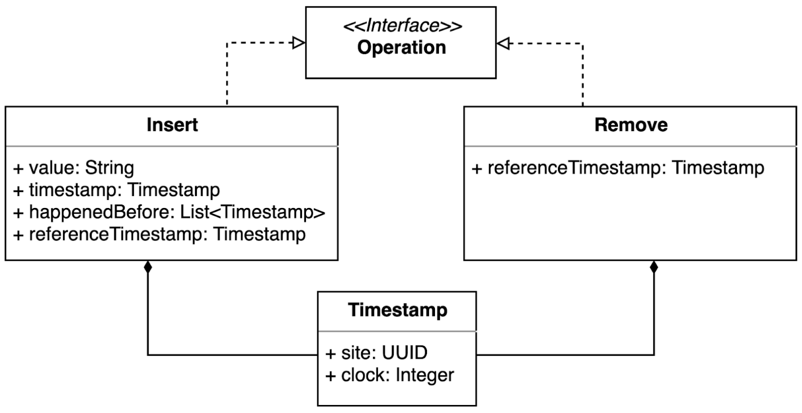


Figure 16. Operation structure.

When remote operations are received, they are processed in the same way as the local ones. In case of remote operations, however, there is no need to search the tree node by text position. With the help of the hash table this process takes effectively constant time.

2.2.11 Causal broadcast

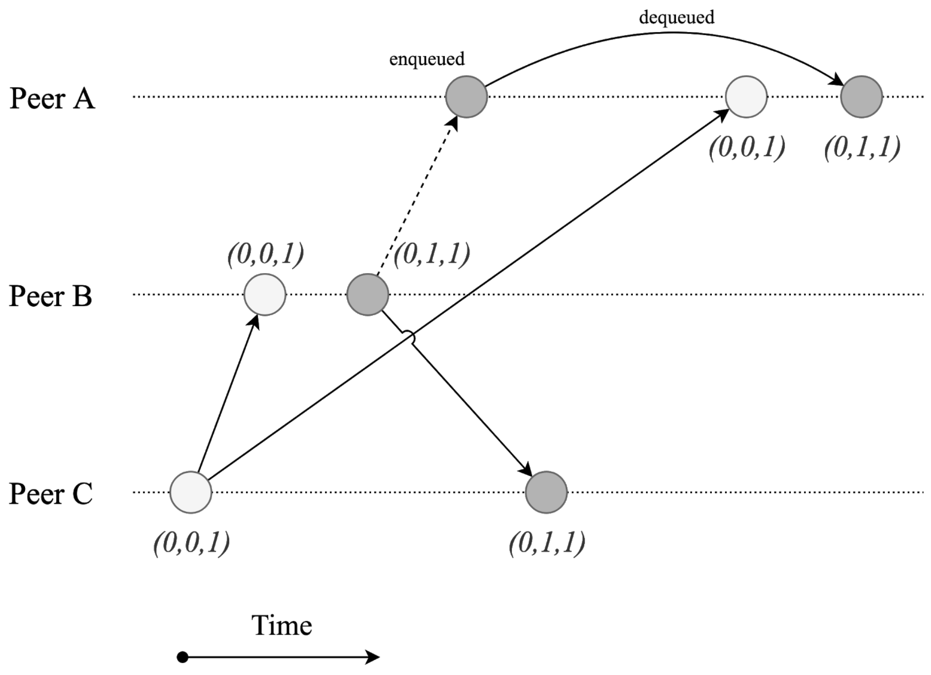


Figure 17. *Birman-Schiper-Stephenson* algorithm. Peer A receives message from Peer B, but it has not observed previous message from Peer C yet. Therefore, this message is queued until preceding message arrives.

In order to support tree-based representation, causal broadcast is implemented at the messaging layer. It is based on the previously described *Birman-Schiper-Stephenson* algorithm:

1. Local vector clock is incremented each time a message is broadcasted. This clock is transmitted together with message.
2. When a message is received, depending on the local vector clock, it is either processed immediately, or stored in priority queue (ordered by vector clock).
3. After processing a message, local vector clock is merged with the message clock. Priority queue is repeatedly polled for messages which are now ready for consumption.

2.3 Description of implemented system

A collaborative text editor was implemented according to the specification.[3] Whole application consists of around 2000 lines of mixed TypeScript and JavaScript code. Modules supporting networking and causal delivery contribute the largest part to the codebase. The core CRDT itself is relatively simpler to implement - around 500 lines of TypeScript code.

For the most part, application interface consists of a single text area, which is augmented with collaborative editing capabilities.

Table 5. Browser combinations matrix of performed manual tests.

Browser	**Chrome (PC)**	**Firefox (PC)**	**Chrome (Android)**
Chrome (PC)	+	+	+
Firefox (PC)	+	+	+
Chrome (Android)	+	+	+

Manual end-to-end tests between different browser vendors were performed. Tested scenarios:

1. Inserting and removing parts of a same sentence from different browsers.
2. Simultaneously inserting text from PC and Android devices.
3. Performing Copy & Paste on large chunks of text.
4. Connecting to an existing document and synchronizing latest changes.

Table 5 shows tested browser combinations. In all testing scenarios application remained responsive, correctly captured user intent and tolerated out-of-order message delivery.

2.3.1 Causal Tree benchmark results

Initial application implementation suffered increasingly slower typing performance as text amount grew in size. To pinpoint the performance bottleneck, a performance testing was conducted.

Tests were performed inside Chrome Developer Tools on a computer with 2.5 GHz Intel Core i7 CPU (256 KB L2 and 6MB L3 caches), 1600 MHz DDR3 memory. Because JavaScript is executed in a single thread inside Google V8 engine, additional CPU cores are not expected to significantly contribute to the performance.

[3] Application source code is available at https://github.com/prSquirrel/crdt-demo

Table 6. Execution time of local operations using naive Causal Tree implementation.

Local	**100 ops**	**1K ops**	**10K ops**
Insert (append)	17.9 ms	34.5 ms	1433.5 ms
Insert (prepend)	13.1 ms	132.0 ms	Out of memory
Insert (random)	18.6 ms	36.6 ms	2438.5 ms
Delete (start)	15.5 ms	43.6 ms	2371.0 ms
Delete (end)	27.8 ms	56.0 ms	2771.2 ms
Delete (random)	14.5 ms	44.8 ms	2547.2 ms

Table 7. Integration time of remote operations using naive Causal Tree implementation.

Remote	**100 ops**	**1K ops**	**10K ops**
Insert (random)	11.2 ms	12.0 ms	30.0 ms
Delete (random)	10.7 ms	11.8 ms	28.1 ms

Table 8. Memory usage of naive Causal Tree implementation.

Local	**100 ops**	**1K ops**	**10K ops**
Retained Size	64 KB	590 KB	6 061 KB

Tables 6 shows execution time growing more than linearly with input size. Inserting characters to the beginning of a document is a curious case - because characters get added to the same root node, each insertion has to capture all observed identifiers of the same parent. As a consequence, larger inputs consumed excessive memory. However, such reverse editing scenario is very unlikely to happen in practice.

Memory usage measurements shown in Table 8 grow linearly with input size, which is expected for CRDTs.

Looking up underlying tree node by the integer position of depth-first traversal was found to be the most time-consuming operation. This conclusion agrees with the results in Table 7 - integrating remote operations only performs a hash table lookup.

Naive approach is to perform such traversal each time an index is requested. However, as discussed before, a better alternative is to keep an additional data structure for quick index lookups.

An optimized application version additionally stores tree node references in a linked-list, backed by an order statistic splay tree[4]. As splay trees are self-balancing, all operations can be performed in amortized $O(log\ n)$ time with minor increase in memory consumption.

Table 9. Execution time of local operations using optimized Causal Tree implementation.

Local	**100 ops**	**1K ops**	**10K ops**	**100K ops**	**1M ops**
Insert (append)	12.2 ms	13.8 ms	40.2 ms	187.2 ms	1797.2 ms
Insert (prepend)	20.0 ms	147.1 ms	Out of memory	Out of memory	Out of memory
Insert (random)	16.2 ms	22.6 ms	89.5 ms	550.1 ms	6996.5 ms
Delete (start)	10.9 ms	12.4 ms	40.6 ms	162.5 ms	1604.0 ms
Delete (end)	11.2 ms	12.4 ms	28.4 ms	139.7 ms	1462.7 ms
Delete (random)	11.8 ms	18.7 ms	59.9 ms	312.6 ms	3790.0 ms

Table 10. Integration time of remote operations using optimized Causal Tree implementation.

Remote	**100 ops**	**1K ops**	**10K ops**	**100K ops**	**1M ops**
Insert (random)	10.7 ms	19.3 ms	47.3 ms	537.2 ms	7387.8 ms
Delete (random)	12.1 ms	16.7 ms	48.0 ms	432.0 ms	5312.1 ms

Table 11. Memory usage of optimized Causal Tree implementation.

Local	**100 ops**	**1K ops**	**10K ops**	**100K ops**	**1M ops**
Retained Size	73 KB	644 KB	6 656 KB	65 843 KB	646 144 KB

[4] https://github.com/davidbau/splaylist

An implementation making use of optimized identifier structure performs orders of magnitude faster for larger inputs, and is able to handle 1 million characters (Tables 9 and 10). For documents beyond that size, memory usage becomes a limiting factor (Table 11).

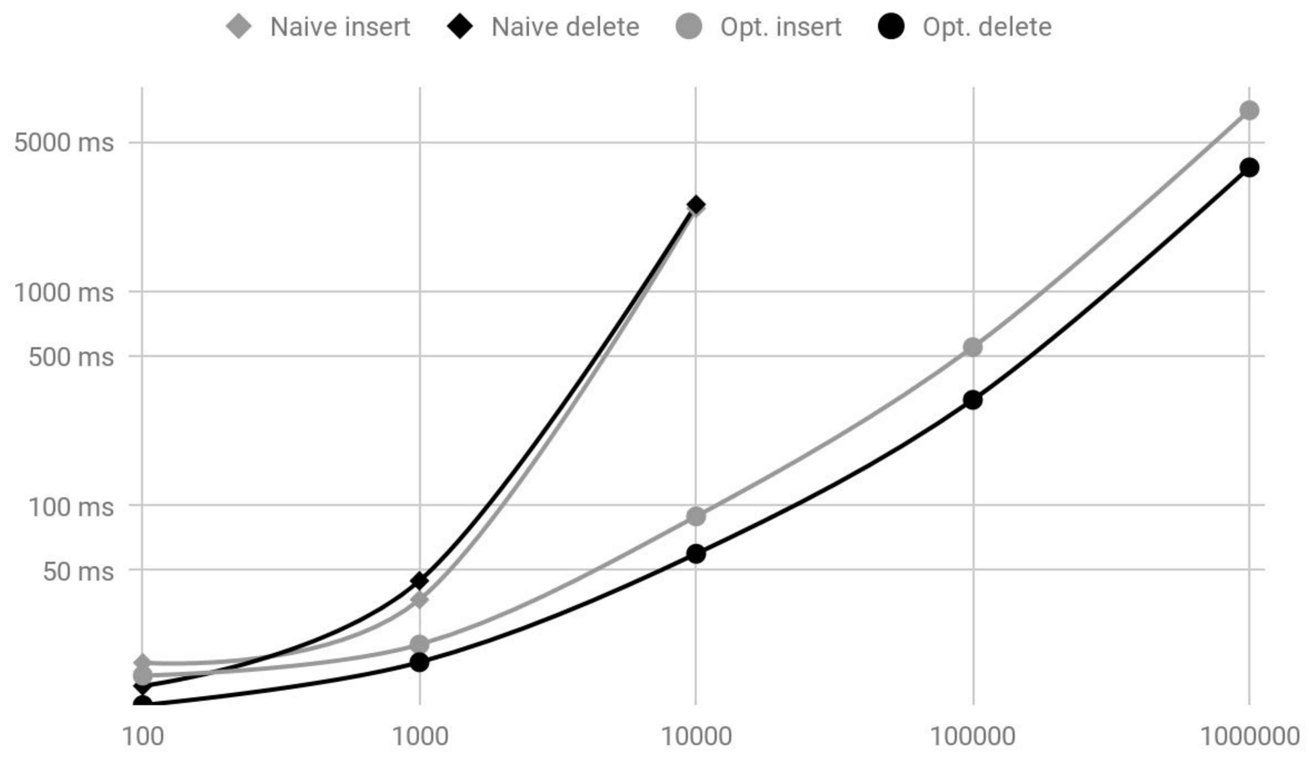


Figure 18. Execution time of naive and optimized Causal Tree implementations. All operations were performed in random order.

A comparison of naive and optimized versions is shown in Figure 18, with a significant performance improvement for local editing scenarios.

2.4 Conclusions

Main architectural points of designing collaborative applications using CRDTs were discussed. Unfortunately, simpler CRDTs such as grow-only sets and counters, while being valuable in databases, are not very useful in collaborative applications. Because of the requirement for user intent preservation and total ordering between elements, a family of algorithms known as Sequence CRDTs are used.

A peer-to-peer collaborative text editor was developed and tested. The application is highly-available and fulfills the requirements of convergence, causality and intent preservation. Performance test results show excellent time complexity on the order of $O(log\ n)$. However, underlying CRDT is linear in memory usage. Practical findings agree closely with theory found in literature.

3 CONCLUSIONS

As stated in CAP theorem, highly-available applications trade availability for consistency. [3] As a result of eventual consistency, some form of conflict resolution is required, such that all replicas agree with each other. Many solutions to conflict resolution exist today, such as timestamping each operation or even delegating it to user. However, collaborative applications require more robust consistency models collectively known as convergence, causality and intent preservation. [24] Two main families of algorithms enabling this model were studied - Operational Transformation (OT) and Conflict-free Replicated Data Types (CRDT).

The nature of distributed systems make it difficult to consider each edge case separately. Furthermore, it becomes even more evident in such extreme cases as full mesh peer-to-peer communication. OT is more mature and well-understood model, however, its time complexity grows with the size of a document and it is quite difficult to implement properly without a central server. In such cases, provable algorithms with a strong mathematical foundation such as CRDTs are preferable.

CRDTs are already battle-tested in such eventually consistent databases as Redis, Dynamo, and Riak by large companies in such industries as online chat systems, gaming, gambling and content delivery systems. Their application in collaborative editing is still relatively new, with many sequence CRDTs showing promising results, even though recently interleaving anomalies were found to be present in some algorithms. [28]

In hindsight, OT handles each conflict explicitly, while CRDT does so implicitly by enforcing local invariants such as commutativity. In general, it is computationally simpler to impose strong local invariants on each replica, which in turn enable global properties across the system. CRDT is a promising field, but for mainstream usage further research is needed on reducing the memory usage of supporting local invariants, and enabling proper garbage collection algorithms for long-running applications.

A peer-to-peer collaborative text editor was designed, implemented and tested. Underlying text is stored in a Causal Tree data structure, which adheres to the sequence CRDT laws. A Causal Tree implementation with applied improvements described by Martin Kleppmann et al. does not suffer from intention-violating interleaving anomalies. [28] Even more, application fulfills consistency model of convergence, causality and intent preservation. It is also fault-tolerant up to loss of $n-1$ nodes in a system.

The developed text editor highlights main strength of CRDTs as opposed to OT - very fast execution on local and remote replicas. In practice, collaborative applications tend to favour operation-based CmRDTs as it is too expensive to replicate full state in real-time. CmRDTs require reliable causally-ordered communication protocol from messaging middleware. Moreover, a single operation applied in the wrong order will corrupt the state, therefore it is very important to verify that CRDT properties hold at all times. This is achieved by performing automated tests (preferably property-based) during development, and input validation during runtime.

When developing collaborative applications, it is crucial to recognize implications of automatic conflict resolution. Consider two persons editing a shared document for days in isolation based on their respective knowledge, and then synchronizing their changes. An argument is started, and two parties arrive at a resolution completely different from an automatic one.

This scenario is especially common in software development world, in which VCS systems such as Git delegate conflict resolution up to the user. Discussed algorithms are capable of capturing very small intention contexts, but would override higher level strategic decisions. Perhaps the more robust solution is to select conflict resolution strategy based on certain heuristics such as time spent editing offline document.

REFERENCES

[1] Ellis, C.A. and Gibbs, S.J., 1989, June. Concurrency control in groupware systems. In Acm Sigmod Record (Vol. 18, No. 2, pp. 399-407). ACM. URL: http://dx.doi.org/10.1145/67544.66963

[2] Li, D. and Li, R., 2010. An admissibility-based operational transformation framework for collaborative editing systems. Computer Supported Cooperative Work (CSCW), 19(1), pp.1-43. URL: https://doi.org/10.1007/s10606-009-9103-1

[3] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. SIGACT News 33, 2 (June 2002), 51-59. URL: https://doi.org/10.1145/564585.564601

[4] Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M., 2011, October. Conflict-free replicated data types. In Symposium on Self-Stabilizing Systems (pp. 386-400). Springer, Berlin, Heidelberg. URL: https://doi.org/10.1007/978-3-642-24550-3_29

[5] Shapiro, M., Preguiça, N., Baquero, C. and Zawirski, M., 2011. A comprehensive study of convergent and commutative replicated data types (Doctoral dissertation, Inria–Centre Paris-Rocquencourt; INRIA). URL: https://hal.inria.fr/inria-00555588

[6] Grudin, J., 1994. Computer-supported cooperative work: History and focus. Computer, 27(5), pp.19-26.

[7] Wilson, P., 1991. Computer supported cooperative work:: An introduction. Springer Science & Business Media.

[8] Robert Johansen. 1988. Groupware: Computer Support for Business Teams. The Free Press, New York, NY, USA.

[9] Bernier, Y.W., 2001, March. Latency compensating methods in client/server in-game protocol design and optimization. In Game Developers Conference (Vol. 98033, No. 425).

[10] Wang, D., Mah, A. and Lassen, S., 2010. Google wave operational transformation. Whitepaper, Google Inc. Online (accessed 2019-03-31) https://svn.apache.org/repos/asf/incubator/wave/whitepapers/operational-transform/operational-transform.html

[11] Daniel, S., 2010. Understanding and Applying Operational Transformation. Online (accessed 2019-03-31). URL:

http://www.codecommit.com/blog/java/understanding-and-applying-operational-transformation

[12] Brewer, E., 2012. CAP Twelve years Later: how the. Computer, (2), pp.23-29. URL: http://alchem.usc.edu/courses-ee599/downloads/T_CO2_CAP12YearsLater.pdf

[13] Abadi, D., 2012. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. Computer, 45(2), pp.37-42. URL: http://md.ssdi.di.fct.unl.pt/resources/Summaries/abadi-pacelc.pdf

[14] Preguiça, N., Baquero, C. and Shapiro, M., 2018. Conflict-free Replicated Data Types (CRDTs). arXiv preprint arXiv:1805.06358. URL: https://arxiv.org/pdf/1805.06358.pdf

[15] Lamport, L., 1978. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7), pp.558-565. URL: https://citemaster.net/get/10b50274-7bc5-11e5-8aa1-00163e009cc7/p558-lamport.pdf

[16] Deutsch, P., 1994. The eight fallacies of distributed computing.

[17] Mills, D., Martin, J., Burbank, J. and Kasch, W., 2010. RFC 5905: Network time protocol version 4: Protocol and algorithms specification. Internet Engineering Task Force. URL: https://www.ietf.org/rfc/rfc5905.txt

[18] Lichvar, M., 2015. Five different ways to handle leap seconds with NTP. Online (accessed 2019-04-07). URL: https://developers.redhat.com/blog/2015/06/01/five-different-ways-handle-leap-seconds-ntp/

[19] Fidge, C.J., 1987. Timestamps in message-passing systems that preserve the partial ordering (pp. 56-66). Australian National University. Department of Computer Science. URL: http://zoo.cs.yale.edu/classes/cs426/2012/lab/bib/fidge88timestamps.pdf

[20] Mattern, F., 1988. Virtual time and global states of distributed systems (pp. 215-226). Univ., Department of Computer Science. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.4399&rep=rep1&type=pdf

[21] Imine, A., Rusinowitch, M., Oster, G. and Molli, P., 2006. Formal design and verification of operational transformation algorithms for copies convergence. Theoretical Computer Science, 351(2), pp.167-183. URL: https://www.sciencedirect.com/science/article/pii/S030439750500616X/pdf?md5=9985c6add113bffafc2c09546f2c176f&pid=1-s2.0-S030439750500616X-main.pdf&_valck=1

[22] Birman, K., Schiper, A. and Stephenson, P., 1991. Lightweight causal and atomic group multicast. ACM transactions on Computer Systems, 9(ARTICLE), pp.272-314. URL: https://infoscience.epfl.ch/record/50197/files/BSS91.pdf

[23] Attiya, H., Burckhardt, S., Gotsman, A., Morrison, A., Yang, H. and Zawirski, M., 2016, July. Specification and complexity of collaborative text editing. In Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (pp. 259-268). ACM. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/podc16-complete.pdf

[24] Sun, C., Jia, X., Zhang, Y., Yang, Y. and Chen, D., 1998. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Transactions on Computer-Human Interaction (TOCHI), 5(1), pp.63-108. URL: http://salvin.jeancharles.free.fr/Documents/Projet%20-%20Boulot/NTU-Singapore/p63-sun.pdf

[25] Ellis, C.A. and Sun, C., 1998, November. Operational transformation in real-time group editors: issues, algorithms, and achievements. In Proceedings of the 1998 ACM conference on Computer supported cooperative work (pp. 59-68). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.933&rep=rep1&type=pdf

[26] Fatin, P., 2015. Typing with pleasure. Online (accessed 2019-05-06). URL: https://pavelfatin.com/typing-with-pleasure/

[27] Attiya, H., Burckhardt, S., Gotsman, A., Morrison, A., Yang, H. and Zawirski, M., 2016, July. Specification and complexity of collaborative text editing. In Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (pp. 259-268). ACM. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/podc16-complete.pdf

[28] Kleppmann, M., Borges Ferreira Gomes, V., Mulligan, D.P. and Beresford, A., 2019. Interleaving anomalies in collaborative text editors. URL: http://martin.kleppmann.com/papers/interleaving-papoc19.pdf

[29] Briot, L., Urso, P. and Shapiro, M., 2016, November. High responsiveness for group editing crdts. In Proceedings of the 19th International Conference on Supporting Group Work (pp. 51-60). ACM. URL: https://hal.inria.fr/hal-01343941/document

[30] MDN Web Docs, 2019. WebRTC Connectivity. Online (accessed 2019-05-14). URL: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity

[31] Grishchenko, V., Causal trees: towards real-time read-write hypertext. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.627.5286&rep=rep1&type=pdf