A Survey of Concurrency Control Algorithms in Collaborative Applications

Sherief Alaa[1], Karim Marzouq[2]

[1]sheriefalaa.w@gmail.com

[2]karim@apkallum.com

***Abstract*—Collaborative applications are becoming more prevalent for a variety of reasons, most important of which is the increased interest in remote work. In addition to adapting the business processes to a remote setting, designers of collaborative software have to decide on how their software can be used collaboratively. This paper discusses the two main technologies used to enable network-based real- or near-real-time collaborative software, namely Operational Transformation and Conflict-free Replicated Data Types. Recent developments in each technology are discussed, as well as a brief overview of their theoretical underpinnings.**

Keywords—Operational Transformation, Conflict-free replicated data type, eventual consistency, OT, CRDT, collaborative text editing, strong eventual consistency.

I. Introduction

Cloud based collaboration software applications (groupware software) are the future of productivity applications. They do not need to be installed on a machine as the browser acts as a client. Moreover, the client can automatically save to the cloud without the need to backup important work, and allow multiple users to edit concurrently (e.g. Google Docs, Microsoft Word Online, etc...)[1], [2].

A groupware software differs from a multi-user software. The former has a shared state which all participants edit concurrently, while in the latter, each user has their own state which is not editable concurrently. To describe a certain software as a groupware software, certain properties must be met before it can be categorized as such: being real-time, having an interactive user interface, being distributed, and access-controlled [3].

Implementing such applications on the web proved to be challenging due to a number of reasons, such as network latency and consistency maintenance [4]. Several methods were introduced in the last three decades to achieve expected user intentions while concurrently editing documents and files. Choosing one over the other is a challenge for technical teams.

In this paper, we introduce and discuss the properties of two consistency control & concurrency maintenance systems. The first system is Operation Transformation (OT) by Ellis and Gibbs. It is a system of operation logs where user actions are transformed to atomic operations which are sent to a central server to transform the shared state [3]. The second system is Conflict-free Replicated Data Type (CRDT). It is a state-based concurrency control system where each participant has a copy of the state and mutates locally, which are mathematically guaranteed to automatically resolve conflicts upon merging [5].

The rest of this paper is organized as follows. 4 sections. Section 2 gives an overview about Operation Transformation (OT) and how it functions using an example. In section 3, we explore the various capabilities of OT. In section 4, we explore CRDTs and give an overview on how it functions. Finally, we conclude in section 5 with a conclusion.

II. Operation Transformation

Operation Transformation (OT) is a system of algorithms first invented in 1989 to achieve concurrency control in collaborative editing software [3].

Collaborative applications in the last decade have adopted a client-server model. The server acts as a central point to host the state of the shared document. Having a central server simplifies implementation,

enables distribution, and facilities fault tolerance. Thus, OT relies on a central server to contain a shared state where multiple users can send their insert and delete operations then the server propagates back those changes to all other connected users. Multiple users in different locations around the world can perform edits concurrently. However, one server is usually not enough to withstand the load. Due to that, a replicated server architecture was introduced. The application and shared documents are replicated at all co-editing servers. Each user can directly edit the replicated document locally and see their edits instantaneously. Local edits are then sent (propagated) to the remote replicated server which is responsible for replaying the changes to other replicas. The replay occurs through a series of *operations* which transform the state of other replicas [6].

A major challenge to OT is how to replay the concurrent operations received by one replica to other replicas and achieve consistency across all replicas. To achieve consistency in a replicated architecture three conditions must be met. The first is *convergence* i.e. all replicas states must be equal after applying a set of operations on them. The second is *causality-preservation*, i.e. all operations must be relayed by their order to satisfy the happened-before relationship in distributed systems [7]. The third is *user intention preservation*, i.e. any side effects of a local operation on a local replica must be propagated to all remote replicas so that all connected clients have the same state [8].

To solve both convergence and intention preservation in OT, the *transformation* function was invented [9]. There are two types of transformation functions as described by Sun et al. [10], the first is *Inclusion Transformation (IT)* and the second is *Exclusion Transformation (ET)*. The former includes data and the latter excludes data from the shared state. Original operations by clients are transformed locally, keeping in mind other concurrent operations, so that executing the new version on a remote replica will have the same effect as executing on the local replica. This method allows concurrent operations to be executed in different order, thus achieving commutativity and intention preservation. For OT to achieve causality preservation, distributed system techniques can be adopted outside of the transformation function, i.e. Lamport's Clock Condition [7]. See Figure 1 for an overview of the lifecycle in an OT system.

Basic OT lifecycle starts out with a document that may or may not contain a string, in the following example the document contains the string “Helloo”. The document is replicated to all connected clients, in this case, User A and User B. Both clients edit the shared document concurrently. The first concurrent local operation is: $U1 = insert(5, \text{“!”})$ to insert the character “!” at the fifth position in the string “Helloo” for it to become “Hello!o”. The second concurrent local operation is: $U2 = delete(5, 1)$ deletes one character starting at the fifth position of the string to become “Hello”. Both $U1$ and $U2$ are executed locally on their respective replicated state and take effect instantaneously.

Afterwards, local concurrent operations $U1$ and $U2$ will be sent to the central server for transformation and replaying the transformed operations to all connected replicas. On the central server, $U2$ will be transformed against $U1$ as: $U2' = transform(U2, U1) = delete(6, 1)$ which will increment the position of the character to be deleted as to preserve User B’s intention. As for U1, it will be transformed as: $U1' = transform(U1, U2) = insert(5, \text{"!"})$ which is identical to the original local concurrent operation $U1$ because it does not conflict with $U2$.

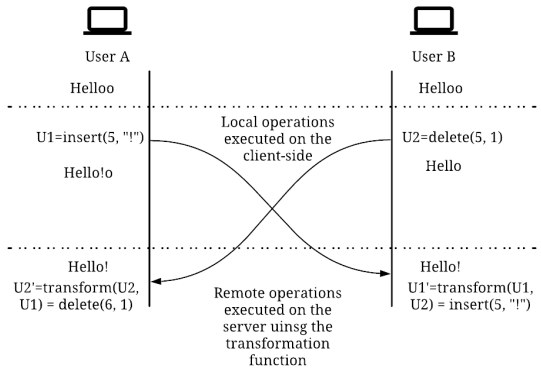

Figure 1 (Typical life-cycle of an OT system transforming operations for consistency maintenance)

III. OT CAPABILITIES

Three decades ago, OT was invented to solve only collaborative editing in documents, with a focus on consistency maintenance and automatic merge conflict resolution (both explained in the previous section). However, research expanded to further areas which enabled more capabilities in collaborative applications, such as operation compression, collaborative graphic design tools, and database synchronization. We will explore the most important OT capabilities that resulted from research in this section.

A. Locking

OT supports optional locking that can be complementary in preserving data integrity. For example, if User1 wrote the sentence "Coffee contain caffeine", then two other users noticed the grammar mistake and decided to solve it. The first may correct it as "Coffee contains caffeine" and the second may correct it as "Coffee can contain caffeine" at the exact same time. The end result would be "Coffee can contains caffeine" which is incorrect. However, with locking, an editor can get a lock on a specific section in the document before modifying it, thus ensuring correct user intention, and excluding potential conflicts [11].

B. Transparent Adaptation

Transparent Adaptation (TA) is a technology that fully depends on OT. TA can take an existing single user application and convert it to a multi-user one without altering the source code of that application. It will also grant all of the OT collaborative capabilities to that application [12]. See Figure 2 that explains the different layers needed to implement TA.

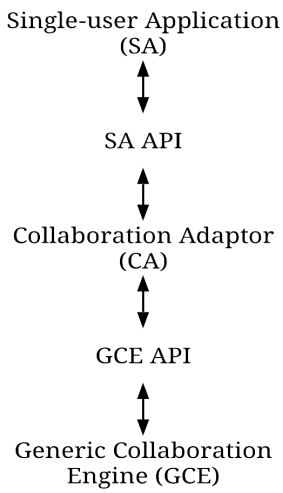

Figure 2 (Basic outline of a TA system)

1. Layer 1: the Single-user Application (SA) possibly any existing single-user application (e.g. Windows NotePad). An SA only provides the user with a user interface for a single-user. It should not complicate the interface or even know there are multiple users editing the document.
2. Layer 2: the Collaboration Adaptor (CA). The CA offers collaborative features that are distinct to each application. The CA communicates with the SA API (Application Programming Interface) to enhance the SA with collaboration features, without altering the source code of the SA. The CA plays a critical role in transforming the SA into a Collaborative Single-user Application (CoSA) by bridging the SA and the underlying OT-powered Generic Collaboration Engine, and in extending generic OT-based collaboration features to support different single-user applications.
3. Layer 3: Generic Collaboration Engine (GCE). The GCE supports OT-powered collaboration features in consistency maintenance, concurrency control, workspace awareness, interaction control, etc. It is a generic layer and can be applied to a wide variety of applications.

Single-user functions (SA) are distinguished from multi-user collaboration capabilities, and application-specific collaboration capabilities (CA) are distinguished from generic collaboration features in the TA reference design (GCE). The job of converting a single-user application is simplified to investigating, designing, and implementing a new CA for this new application using this architecture and the reusable GCE component [12], [13].

C. *Group undo*

Undo operations are necessary to ensure collaborative software reliability. Users should not abstain from using collaborative software because they can not expect the outcome of an undo operation. Several attempts were made in the last two decades to perfect the undo operation. The first method introduced is called DistEdit [14], it allows undo operations back and forth. However, if the undo operation was conflicting with another operation, it becomes undoable. The second approach is called adOPTed, it is a system where each user can undo all their operations as long as it's their operations only and using OT. The third method is called REDUCE [15], it also uses OT and its time complexity is *O(N)*.

IV. Conflict-Free Replicated Data Type

Owing to the rise in popularity of collaborative software, further research into algorithms and collaborative editing led to the emergence of Conflict-free Replicated Data Types (CRDT). OTs are often described as an "incorrect, complex and inefficient technique" [8]. This is due to its complexity and edge cases — some OT implementations have been formally proven to diverge after a valid sequence of operations, while others evade attempts of formal proof due to the sheer complexity of the implementation [16]. As mentioned earlier in this paper, OT requires a central server to function, which requires at minimum, trusting the central server (see Figure 3). Thus, another major reason for research into CRDTs is their support for decentralization, i.e: their ability to operate without a central server. The nature of a globally connected network of modern clouds means that two users sitting in the same room, working concurrently on the same file, are required to utilize the internet to propagate changes amongst themselves. See Figure 4.

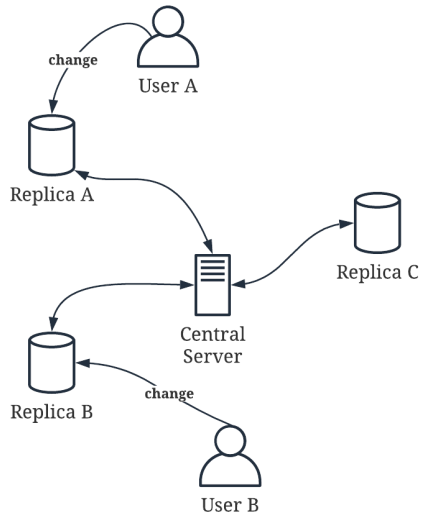

Figure 3 (Typical OT network topology illustration)

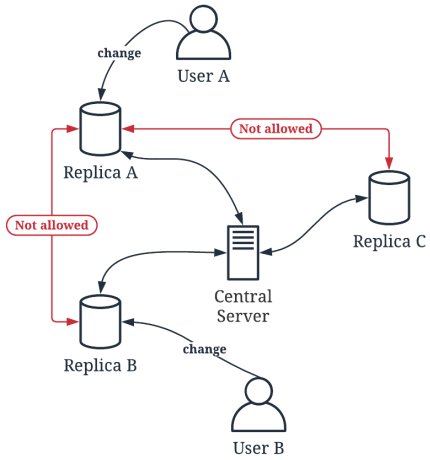

Figure 4 (Inflexibility of node-to-node communication in an OT network topology illustration)

CRDTs guarantee that all data replicas converge towards a deterministic value when merged [17]. The CAP theorem states that designers of distributed systems have to contend with favoring either strong consistency or availability in order to deal with network partitioning [18]. Consistency means guaranteeing that each read receives the most recent write or an error. Availability means that each read returns a response, which could be stale, but not an

error. Since network faults are a built-in assumption in any internet-based environment, and given the nature of most internet applications, availability is usually favored over strong consistency [19].

A. *Strong Consistency & Eventual Consistency*

A distributed system is said to be strongly consistent if all of its nodes see all requests coming in sequentially. This means that for any given deterministic object, all nodes in a strongly consistent system will see the same sequence of write operations. This strong consistency offers non-byzantine fault-tolerance, at the expense of adding a bottleneck to the system. This bottleneck represents the increased latency it takes to propagate changes among the nodes. This latency includes both network latency and the time it takes to arrive at a consensus among the nodes [20].

Such consensus algorithms and protocols, for example Paxos [21] or Raft [22]. Paxos guarantees that a strong consistent state is reached among nodes in a partitioned network by picking one value among several proposed values. On the other hand, Raft archives a strong consistent state by electing a leader node that handles and broadcasts all writes.

On the other hand, eventual consistency prioritizes availability of nodes by accepting write or update requests coming to any node. An eventually consistent system will respond to any read — regardless of the node — with the data this node contains. However, there are no guarantees that it is the most up-to-date data. It is worth mentioning again that this is usually the priority for real-world systems — consistency anomalies are usually dealt with by external compensation mechanisms. Even bank accounts, the most popular example to illustrate the need for strong consistency, are usually implemented with only eventual consistency guarantees [1] [19].

However, the needs of users of collaborative, groupware applications are orthogonal to the consensus algorithm approach. This is because in collaborative software, the merging of writes sensibly is the desired behavior — no writes to any node should be discarded. Additionally, a network partition constraint is a valid assumption in collaborative applications, since users go offline and expect to propagate their changes upon reconnection. Moreover, the user interface should expose a tentative state such that the user can interact with their data while it's being synced in the background. Those requirements necessitated more research into other collaboration algorithms.

B. *Total order, partial order, and join semilattice*

To explain how CRDTs came into existence, we must first take a look at lattice theory. In the broadest sense, order is a formalization of the general idea of comparison over elements of a set. To create an order, we have to first define a binary (dyadic) relationship, which by definition can be applied to any two elements of a set. For a set of natural numbers S, we define the binary relation $\leq$ to mean less than or equal. Thus we can say that $3 \leq 7$, and $4 \leq 7$, thereby creating an order. Since all elements of the set S are comparable, i.e we can apply the operant $\leq$ to any two elements of the set, it can be said that the elements of the set S are in total order [25].

Given a second set T, the operant $\leq$ could be defined as 'descendent-of'. In this case, we can say $Son \leq Father$ and $Daughter \leq Father$. However, $Daughter$ and Son would be *incomparable* given that the operant $\leq$ denotes an ancestry relationship. We denote that incomparability by writing $Son \parallel Daughter$. Those elements are part of a partially-ordered set, or a poset.

A second operation that can be applied to elements of a set is the join operation. Given two elements a, b of set S, we can apply the join operant $\vee$ if there is at least one supremum (least upper bound, LUB), defined as an element of S that is greater than or equal to a, b. For example, in the poset T containing elements $\{Cairo, Egypt, Nairobi, Kenya, Africa, Tokyo, Japan, Asia\}$, the result of $Cairo \vee Nairobi$ is equal to $Africa$, since it's the least upper bound to those two elements. See Figure 5.

[1] Blockchain networks are another proliferating technology that achieves strong consistency through consensus based on proof-of-work (or more recently proof-of-stake) [23]. Blockchain networks also add the dimension of trust to the CAP theorem [24].

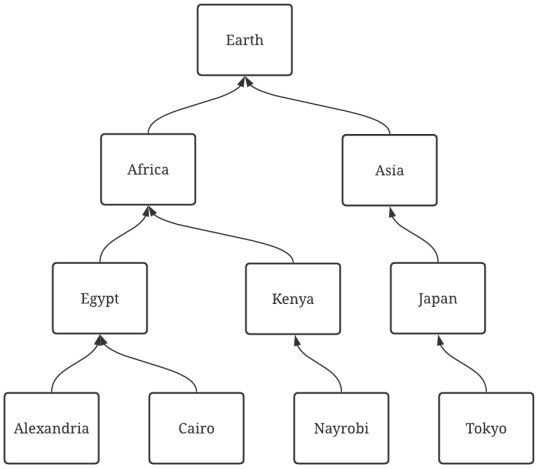

Figure 5 (Least Upper Bound example)

Thus, the join of elements a, b (written $a \vee b$) in the order $(S, \leq)$, is the least upper bound of S according to the defined order. In practice, this means that we are looking for the smallest element x which satisfies $a \leq x$ and $b \leq x$. In the case of total order, i.e: when each pair of elements in the set are comparable to one another, the result of joins is the larger element of a, b.

A join semilattice is an order $(S, \leq)$ such that for each $a, b \in S$ there exists a join $a \vee b$. Such joins obey three laws:

1) Commutativity: $a \vee b = b \vee a$
2) Associativity: $(a \vee b) \vee c = a \vee (b \vee c)$
3) Idempotence: $a \vee a = a$

Conflict is defined by Shapiro et al as "a combination of concurrent updates which may be individually correct, but that, taken together, would violate some invariant." [17]. The aforementioned three properties are what enables us to create CRDTs: *Conflict*-Free Replicated Data Types, which, when merged, converge towards the same result, without the need for resolving conflicts.

C. Vector Clocks & Grow-only Counter

Grow-only Counter CRDT, also known as GCounter, is a counter whose value can only increase. All possible values of the counter, or its states, are considered elements of a set. For that state to be successfully replicated using CRDTs, we must be able to create an order of that set using a binary relation, which is happened-before [7]. We also need a way to join any two elements of that set-state, for our purposes we call that the $merge()$ function.

Building on the work of Lamport's logical clocks, vector clocks were developed to capture casualty among events in a distributed system [20]. A vector timestamp consists of integers of logical timestamps, where each element represents a process or a replica, for example $(2, 5, 8)$. We can define a "happened before" comparison relationship $\leq$ (also denoted as $\rightarrow$) among vector timestamps if every element of vector timestamp $v1$ is less than or equal the corresponding element in vector timestamp $v2$.

As shown in diagram x, the partial and causal order of the system are preserved: at any given time, we can define a happened-before relationship. A system can also be viewed as a join semilattice, visualized in Figure 6:

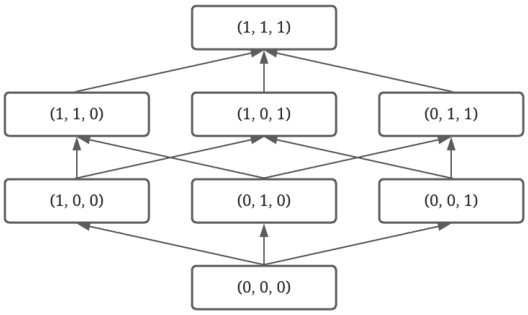

Figure 6 (Join semilattice example)

Our G-Counter can be modeled as a system of 3 replicas. Each replica implements two functions: $increment()$, which increases the counter in a given replica by 1, and $value()$, which returns the value of the global counter, i.e: our system. A third function, $merge(state_{incoming})$. Figure 7 shows the state of the system:

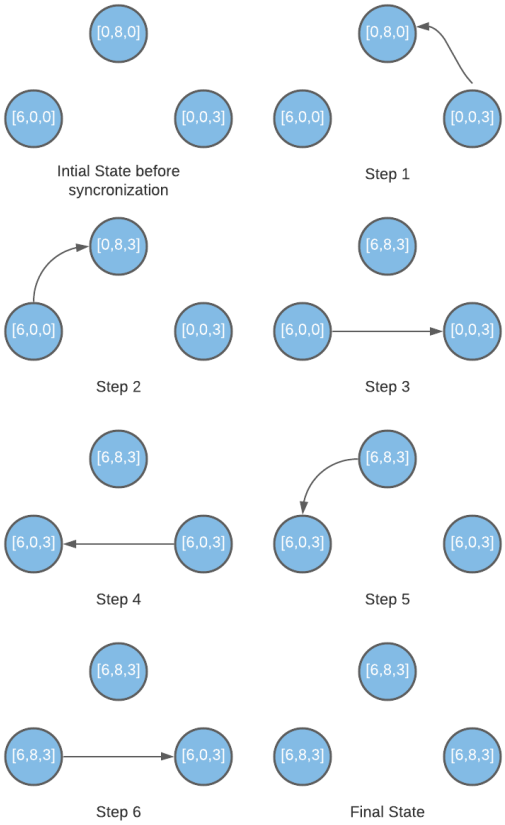

Figure 7 (Life-cycle of a GCounter CRDT)

At first glance it seems that each replica can pass its value to another replica, then change its value to the summation of the two values. This is bound to fail, since it violates the principle of idempotency. Each number in a circle represents the number of times the function $increment()$ was called on a given replica. Thus the total value of our system should converge to 17, since that's the sum of [3, 6, 8]. Thus we can represent the state of each node as an element in an integer vector [26].

In this case, our $merge()$ function takes in a vector of values, of which each element represents the value of a replica, and merges them by taking the highest value for each element. The $value()$ function, which returns the value of the whole system, would then be implemented as the sum of all elements in the state vector.

V. Conclusion

In conclusion, Operation Transformation algorithms have served their purpose for the last three decades. However, their increasing implementation complexity and inflexibility with respect to decentralized network topologies have rendered them less than desirable for modern applications. On the other hand, interest in decentralized, peer-to-peer applications have spurred new research into collaborative technologies. Conflict-free Replicated Data Types are rapidly gaining popularity as collaborative and local-first applications supplant traditional web applications.

Since collaborative text editing is the most ubiquitous collaborative application, many CRDTs focus on modeling text editing as its basis. To that end, the software libraries automerge and yjs have emerged as valid open-source solutions for building collaborative web applications. Both libraries are written in JavaScript and use JSON as the data representation layer, thus making them versatile in different environments, with emphasis on web browsers. The strong mathematical underpinnings of CRDTs, along with their relative ease of implementation to OT, have made them a strong contender to dominate the collaborative application space. Future research should be undertaken to further integrate CRDTs into real-world systems.

References

[1] E. Liu, "A CRDT-based file synchronization system," 2021, Accessed: Nov. 06, 2021. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2778095

[2] J. Harris, "What's different about the new Google Docs?," *Google Drive Blog*, May 11, 2010. https://drive.googleblog.com/2010/05/whats-different-about-new-google-docs.html (accessed Nov. 27, 2021).

[3] C. A. Ellis and S. J. Gibbs, "Concurrency control in groupware systems," in *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, New York, NY, USA, Jun. 1989, pp. 399–407. doi: 10.1145/67544.66963.

[4] C. Sun and C. Ellis, “Operational transformation in real-time group editors: issues, algorithms, and achievements,” in *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, New York, NY, USA, Nov. 1998, pp. 59–68. doi: 10.1145/289444.289469.

[5] N. Preguica, J. M. Marques, M. Shapiro, and M. Letia, “A Commutative Replicated Data Type for Cooperative Editing,” in *2009 29th IEEE International Conference on Distributed Computing Systems*, Montreal, Quebec, Canada, Jun. 2009, pp. 395–403. doi: 10.1109/ICDCS.2009.20.

[6] S. Weiss, P. Urso, and P. Molli, “Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks,” in *2009 29th IEEE International Conference on Distributed Computing Systems*, Jun. 2009, pp. 404–412. doi: 10.1109/ICDCS.2009.75.

[7] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Commun. ACM*, vol. 21, no. 7, pp. 558–565, Jul. 1978, doi: 10.1145/359545.359563.

[8] C. Sun, D. Sun, A. Ng, W. Cai, and B. Cho, “Real Differences between OT and CRDT under a General Transformation Framework for Consistency Maintenance in Co-Editors,” *Proc. ACM Hum.-Comput. Interact.*, vol. 4, no. GROUP, Art. no. GROUP, Jan. 2020, doi: 10.1145/3375186.

[9] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping, “High-latency, low-bandwidth windowing in the Jupiter collaboration system,” in *Proceedings of the 8th annual ACM symposium on User interface and software technology*, New York, NY, USA, Dec. 1995, pp. 111–120. doi: 10.1145/215585.215706.

[10] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, “Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems,” *ACM Trans. Comput.-Hum. Interact.*, vol. 5, no. 1, pp. 63–108, Mar. 1998, doi: 10.1145/274444.274447.

[11] C. Sun and R. Sosič, “Optimal locking integrated with operational transformation in distributed real-time group editors,” in *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, New York, NY, USA, May 1999, pp. 43–52. doi: 10.1145/301308.301322.

[12] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai, “Transparent adaptation of single-user applications for multi-user real-time collaboration,” *ACM Trans. Comput.-Hum. Interact.*, vol. 13, no. 4, pp. 531–582, Dec. 2006, doi: 10.1145/1188816.1188821.

[13] Agustina, F. Liu, S. Xia, H. Shen, and C. Sun, “CoMaya: incorporating advanced collaboration capabilities into 3d digital media design tools,” in *Proceedings of the 2008 ACM conference on Computer supported cooperative work*, New York, NY, USA, Nov. 2008, pp. 5–8. doi: 10.1145/1460563.1460566.

[14] A. Prakash and M. J. Knister, “A framework for undoing actions in collaborative systems,” *ACM Trans. Comput.-Hum. Interact.*, vol. 1, no. 4, pp. 295–330, Dec. 1994, doi: 10.1145/198425.198427.

[15] C. Sun, “Undo any operation at any time in group editors,” in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, New York, NY, USA, Dec. 2000, pp. 191–200. doi: 10.1145/358916.358990.

[16] D. Li and R. Li, “An Admissibility-Based Operational Transformation Framework for Collaborative Editing Systems,” *Comput. Support. Coop. Work*, vol. 19, no. 1, pp. 1–43, Feb. 2010, doi: 10.1007/s10606-009-9103-1.

[17] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, “Conflict-free Replicated Data Types,” report, Jul. 2011. Accessed: Jan. 08, 2022. [Online]. Available: https://hal.inria.fr/inria-00609399

[18] M. Kleppmann, *Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems*, First edition. Boston: O’Reilly Media, 2017.

[19] P. Bailis and A. Ghodsi, “Eventual Consistency Today: Limitations, Extensions, and Beyond: How can applications be built on eventually consistent infrastructure given no guarantee of safety?,” *Queue*, vol. 11, no. 3, pp. 20–32, Mar. 2013, doi: 10.1145/2460276.2462076.

[20] M. van Steen and A. S. Tanenbaum, *Distributed systems*, Third edition, Version 3.01. Erscheinungsort nicht ermittelbar: Maarten van Steen, 2017.

[21] L. Lamport, D. Malkhi, and L. Zhou, “Vertical paxos and primary-backup replication,” in *Proceedings of the 28th ACM symposium on Principles of distributed computing - PODC ’09*, Calgary, AB, Canada, 2009, p. 312. doi: 10.1145/1582716.1582783.

[22] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, USA, Jun. 2014, pp. 305–320.

[23] F. Casino, T. K. Dasaklis, and C. Patsakis, “A systematic literature review of blockchain-based applications: Current status, classification and

open issues," *Telemat. Inform.*, vol. 36, pp. 55–81, Mar. 2019, doi: 10.1016/j.tele.2018.11.006.

[24] K. Finlow-Bates, "Adding Trust to CAP: Blockchain as a Strong Eventual Consistency Recovery Strategy," Sep. 2017, p. 12. [Online]. Available: https://www.chainfrog.com/wp-content/uploads/2017/09/CAP-paper.pdf

[25] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge: Cambridge University Press, 2002. Accessed: Jan. 08, 2022. [Online]. Available: https://doi.org/10.1017/CBO9780511809088

[26] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of Convergent and Commutative Replicated Data Types," report, Inria – Centre Paris-Rocquencourt ; INRIA, 2011. Accessed: Jan. 14, 2022. [Online]. Available: https://hal.inria.fr/inria-00555588