SAPIENZA
UNIVERSITÀ DI ROMA

Building a Local-First Application Using Conflict-free Replicated Data Types

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Master's Degree in Engineering in Computer Science

Giorgio Francescone
ID number 1974461

Thesis Advisor
Prof. Andrea Vitaletti

Academic Year 2024/2025

Thesis not yet defended

Building a Local-First Application Using Conflict-free Replicated Data Types

Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: francescone.1974461@studenti.uniroma1.it

Abstract

TBD.

Contents

CHAPTER 1

Introduction

The modern computing landscape has moved towards building distributed systems as a means to fulfill the necessity of delivering high volumes of data to a continuously growing number of users over the Internet. It has become commonplace to have systems span over a network (cluster) of machines – or *nodes* – which communicate with each other in a coordinated fashion. Compared to monolithic systems that assign the entire workload to a single machine, this approach efficiently addresses three fundamental properties that the majority of modern systems seeks to achieve.

- *Scalability*: the system's ability to adapt to the scale of the workload by either allocating a larger quantity of resources on one node – *vertical scaling* – or dividing it among different nodes – *horizontal scaling*;
- *Resiliency*: the system's ability to tolerate eventual faults that may affect the correct operation of one or more nodes;
- *Efficiency*: the system's responsiveness expressed in terms of time and space [1].

Advancements in this field sought to optimally achieve these pivotal demands have led to the emergence of more advanced paradigms, such as *cloud computing*, which poses itself as a model that enables ubiquitous, on-demand access to a shared pool of configurable computing resources – networks, servers, storage, services – provisioned with minimal management

[1] An ideal system operates under low latency and high throughput

effort [1]. The advent of cloud computing has fueled an increasing interest in applications that are built to empower collaborative tasks. Platforms like Google Drive[2], Microsoft OneDrive[3], and Dropbox[4] enable groups of two or more users to engage in common tasks by sharing access to a set of resources – documents, spreadsheets, presentations, images, ... – from anywhere in the world, in real time [2].

Within collaborative applications lies the challenge of making sure that concurrent updates are applied seamlessly and without conflicts, especially in situations where users may be working across different geographical locations with varying network latencies. Despite the risk of network connectivity being intermittent, users expect their collaborative experience to be responsive, their changes to be preserved, and conflicts to be resolved seamlessly without loss of data. This challenge can be broadly addressed either through *consensus* – leveraging algorithms such as Paxos or Raft – or through *convergence* [3]. Traditional approaches to achieve convergence, like the one adopted by Google Drive, heavily rely on centralized conflict resolution strategies such as *Operational Transformation* (OT), where a single node acts as the authoritative source of truth and mediates all concurrent changes [4]. While this mechanism has proven to be reliable, a compelling alternative could be achieved by employing a truly decentralized, peer-to-peer (P2P) network, over which data is replicated across multiple nodes and concurrent changes are handled without the need of a centralized authority.

Conflict-free Replicated Data Types (CRDTs) offer a powerful solution suitable to support highly available file sharing and collaborative systems. CRDTs are data structures that allow the system to update any replica independently and concurrently, without coordinating with other replicas [5]. The core benefit is that the data type itself contains an algorithm that automatically resolves any inconsistencies, guaranteeing that replicas will eventually converge to an identical state, even in the case in which a node may receive

[2]https://drive.google.com
[3]https://onedrive.live.com
[4]https://dropbox.com

updates out of order.

The formalized problem addressed by this thesis is the design and implementation of a Proof-of-Concept (PoC) replicated file storage solution that leverages CRDTs to synchronize file contents and metadata. The system is conceptualized as a P2P network where each participating node maintains a local replica of the storage system. All changes made on any node are propagated to other nodes and merged automatically using CRDT algorithms. For the PoC, the complexities of a true P2P network are abstracted away by using a relay server. This server acts as a message broker, broadcasting updates from one node to all other connected nodes, thereby simulating the propagation of changes in a P2P environment. The core of the system is built using the *PyCRDT library*[5], a Python binding for the popular *Yjs*[6] CRDT framework. This allows for the use of mature and well-tested CRDT implementations to model the file system's directory structure, file contents, and metadata. The system is complemented by a simple web-based client that provides a user interface for seamless file management, where users can upload their files and enforce a robust data integrity guarantee by attaching a synthetic digital signature to each file. Other users can validate such signatures through an offline-first validation mechanism, using the data that was received through CRDTs.

The rest of this thesis is structured as follows:

- In **Chapter 2**, we provide a comprehensive overview of the foundational concepts of CRDTs, along with their taxonomy, core data structures, and conflict resolution strategies.
- In **Chapter 3**, we analyze existing CRDT implementations and frameworks, with a particular focus on the Yjs ecosystem, which forms the basis of the PoC.
- In **Chapter 4**, we delve into the design and implementation process a

[5]https://github.com/y-crdt/pycrdt
[6]https://yjs.dev/

functioning PoC that demonstrates the feasibility of using CRDTs for file synchronization in a simulated P2P environment.

- In **Chapter 5**, we evaluate the PoC's performance, identify its strengths and limitations, and propose directions for future work.

CHAPTER 2

Fundamentals of Conflict-free Replicated Data Types

2.1 Consistency in Distributed Systems

Since the early days of networked computing systems, the challenge of maintaining strong consistency (or *linearizability* [6]) has been a critical research topic. As distributed systems started spanning across an ever increasing amount of devices spread over vast geographic locations, ensuring that all the nodes of a system operate on a coherent shared state at all times becomes a nontrivial task. The complexity also lies in making sure that these systems are always available and robust to any fault or partition that the network could be affected to. While traditional approaches to replication, such as Lamport's state machine [7], achieve consistency by relying heavily on central coordination mechanisms, the adoption of such approaches often comes at the cost of higher latency and disruption of availability under network partitions. This trade-off has been formalized in literature by the CAP Theorem [8, 9], which identifies three key guarantees sought by distributed systems:

- *Consistency* (C): all nodes of the systems are up to date with the updates issued to the internal state. In the context of an atomic read/write register, every read operation receives the most recent write – or an error.

- *Availability* (A): every request received by a non-failing node must result in a response.
- *Partition tolerance* (P): the system continues to operate despite faulty conditions, such as network outages, that lead to arbitrary message loss between nodes.

Moreover, the CAP Theorem states that it is impossible for a real-world system to simultaneously guarantee more than two of these three properties.

Since network partitions may always occur, system designers were forced to make explicit choices about the system's behavior during partitions, by either sacrificing consistency or availability. As a consequence, two dominant design philosophies emerged, being AP (Availability-Partition tolerance) and CP (Consistency-Partition tolerance) systems. In AP systems, such as Amazon's Dynamo [10], eventual inconsistencies between replicas that may arise during network outages are tolerated to ensure that the system is kept operational at all times. Conversely, CP systems – such as Google's Bigtable [11] – prioritize strong consistency guarantees, and particular measures are adopted to ensure that inconsistent data is never served, such as affecting the system's availability during partitions. While replicated systems are geared towards the latter approach, high availability needs remain interesting from a business standpoint, as "even the slightest outage has significant financial consequences and impacts customer trust" [10]. For this reason, research was driven towards alternative consistency models, namely those where the consistency guarantees are relaxed, so that all three properties denoted by the CAP Theorem could simultaneously co-exist.

One such consistency model, known as *optimistic replication*, addresses concurrency control among replicas by letting the data be accessed "optimistically" (without a priori synchronization), propagating the resulting updates in the background, and fixing conflicts after they happen [12]. Another prominent example of weak consistency model, named *eventual consistency* (EC), allows for replicas to temporarily diverge, but guarantees

that, when updates to the replicas are no longer issued, they will converge to an identical state [13]. This model significantly enhances availability and responsiveness, as operations can proceed without waiting for acknowledgement from other peers. However, if concurrent updates are not reconciled in a principled manner, replicas may end up in an inconsistent state, leading to erroneous behavior within the system. An alternative approach to conflict resolution has led to the emergence of CRDTs, which provide a mathematically sound framework for achieving state convergence without the need for synchronization among replicas [14].

2.2 Foundational Concepts on Conflict-free Replicated Data Types

Conflict-free Replicated Data Types (CRDTs) are abstract data types specifically engineered for replication across multiple processes in a distributed system. A defining characteristic of CRDTs is that any replica can be modified without requiring immediate coordination with other replicas [15]. This property is fundamental to achieving high availability and low latency in distributed applications.

2.2.1 CRDT Operation Semantics

A CRDT can be implemented as a standard data type, which provides a set of operations, which usually can be distinguished into two categories [14]:

- *query* operations, which perform a lookup on the internal state – without altering it – and return a result based on what is observed;
- *update* operations, which perform a set of actions that explicitly mutate the internal state.

As we will see when discussing CRDT implementations, performing an update on a CRDT which is replicated across n nodes firstly involves updating the state of a single replica – the one which is directly accessed by the

application – and then propagating the update to all other nodes of the network. When nodes receive an update, they apply it to their replica, and autonomously resolve any conflicts that may arise – in other words, they *merge* the update with their internal state. Broadly speaking, designing an effective update operation can be regarded as a complex endeavor, as failure to properly address the correctness criteria involving the set of operations that form the merged update may result in compromising the system's convergence guarantees. More precisely, the implementation of update operations must guarantee convergence by satisfying one or more of the following key properties:

- *commutativity* – the order in which delivered updates are applied to the replica does not affect the final state.
- *associativity* –
- *idempotency* –

In most cases, it is desirable to deal with commutative operations; due to the reliability of the network, a replica may receive updates out of order, and still be required to converge to a state that is consistent with all other replicas, once all the updates are applied.

2.2.2 Strong Eventual Consistency

Convergence in the case of CRDTs is backed by a stronger version of the EC model, known as *Strong Eventual Consistency* (SEC)[15] , in which two replicas reach the same final state if they have received the same set of updates.

Eventual Consistency (EC) An object is said to be *Eventually Consistent* if it satisfies the following properties:

- **Eventual delivery:** An update delivered to a replica is eventually delivered to all replicas;

- **Convergence:** Replicas that have delivered the same updates eventually converge to an equivalent state;
- **Termination:** All method executions terminate.

Strong Eventual Consistency (SEC) An object is said to be *Strongly Eventually Consistent* if it is Eventually Consistent if it satisfies the following additional property:

- **Strong Convergence:** Replicas that have delivered the same updates converge to an equivalent state.

SEC achieves state convergence without requiring complex, ad-hoc conflict resolution logic to be implemented by the developer. Instead, the "conflict-free" nature is embedded within the design of the data type itself; concurrent operations are either inherently commutative or are resolved by a deterministic merge procedure built into the CRDT.

2.2.3 Mathematical foundations

2.3 State-based vs. Operation-based CRDTs

The design and implementation of CRDTs primarilys follow two distinct strategies for replication: *state-based* and *operation-based* [16].

2.3.1 State-based CRDTs (CvRDTs)

State-based CRDTs, also known as convergent replicated data types (CvRDTs), operate on the principle of replicas exchanging their entire current state. When an update to a replica's state is issued, the replica first applies the update locally, and then broadcasts the resulting state to the other replicas. The replicas receiving the updated state employ a *merge* function to combine the received state with their own local state. for convergence to be guaranteed, this merge function must possess three key mathematical properties:

- **Commutativity:** the order in which states are merged does not affect the outcome;

$$x \cdot y = y \cdot x \tag{2.1}$$

- **Associativity:** the order in which merges are grouped does not affect the outcome;

$$(x \cdot y) \cdot z = x \cdot (y \cdot z) \tag{2.2}$$

- **Idempotency:** merging identical states produces the same resulting state (even when the merge is repeated multiple times).

$$x \cdot x = x \tag{2.3}$$

Thanks to these properties, state-based CRDTs can achieve state convergence among replicas, as long as the replicas have received the same set of updates.

2.3.2 Operation-based CRDTs (CmRDTs)

Operation-based CRDTs, also known as commutative replicated data types (CmRDTs), take a different approach. Instead of transmitting the entire updated state, replicas broadcast the update *operations* themselves as they occur locally. These operations are then applied locally by the receiving replicas. For CmRDTs to ensure convergence, the operations must be designed to be commutative when applied concurrently. Furthermore, CmRDTs typically impose stricter constraints on the underlying communication infrastructure. Guarantees such as exactly-once delivery and causal delivery of operations are often necessary to ensure that all operations are applied consistently across replicas.

2.3.3 Delta-state CRDTs

A significant optimization, particularly for state-based CRDTs, is the concept of delta-state CRDTs (δ-$CRDTs$). Instead of transmitting the entire state, δ-CRDTs only need to transmit the incremental changes (or *deltas*) that have occurred since the last state update. These deltas are then merged at the receiving replicas. This approach overcomes the limitations of state-based replication, which can often be computationally expensive if the state object to transmit is large in size.

Bibliography

[1] Mohammad Hamdaqa and Ladan Tahvildari. "Cloud Computing Uncovered: A Research Landscape". In: *Advances in Computers* 86 (Dec. 2012). DOI: 10.1016/B978-0-12-396535-6.00002-8.

[2] C. A. Ellis and S. J. Gibbs. "Concurrency control in groupware systems". In: *SIGMOD Rec.* 18.2 (June 1989), pp. 399–407. ISSN: 0163-5808. DOI: 10.1145/66926.66963.

[3] Martin Kleppmann and Peter Alvaro. "Research for practice: convergence". In: *Commun. ACM* 65.11 (Oct. 2022), pp. 104–106. ISSN: 0001-0782. DOI: 10.1145/3563901. URL: https://doi.org/10.1145/3563901.

[4] Chengzheng Sun and Clarence Ellis. "Operational transformation in real-time group editors: issues, algorithms, and achievements". In: *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*. CSCW '98. Seattle, Washington, USA: Association for Computing Machinery, 1998, pp. 59–68. ISBN: 1581130090. DOI: 10.1145/289444.289469.

[5] Paulo Sérgio Almeida. "Approaches to Conflict-free Replicated Data Types". In: *ACM Comput. Surv.* 57.2 (Nov. 2024). ISSN: 0360-0300. DOI: 10.1145/3695249.

[6] Maurice P. Herlihy and Jeannette M. Wing. "Linearizability: a correctness condition for concurrent objects". In: *ACM Trans. Program. Lang. Syst.* 12.3 (July 1990), pp. 463–492. ISSN: 0164-0925. DOI: 10.1145/78969.78972.

[7] Leslie Lamport. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems". In: *ACM Transactions on Programming Lan-*

guages and Systems (Apr. 1984), pp. 254–280. DOI: 10.1145/2993.2994.

[8] Eric A. Brewer. "Towards robust distributed systems (abstract)". In: *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '00. Portland, Oregon, USA: Association for Computing Machinery, 2000, p. 7. ISBN: 1581131836. DOI: 10.1145/343477.343502.

[9] Seth Gilbert and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In: *SIGACT News* 33.2 (June 2002), pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601.

[10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. "Dynamo: amazon's highly available key-value store". In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220. ISSN: 0163-5980. DOI: 10.1145/1323293.1294281.

[11] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (June 2008). ISSN: 0734-2071. DOI: 10.1145/1365815.1365816.

[12] Yasushi Saito and Marc Shapiro. "Optimistic replication". In: *ACM Comput. Surv.* 37.1 (Mar. 2005), pp. 42–81. ISSN: 0360-0300. DOI: 10.1145/1057977.1057980.

[13] D.B. Terry, A.J. Demers, K. Petersen, M.J. Spreitzer, M.M. Theimer, and B.B. Welch. "Session guarantees for weakly consistent replicated data". In: *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*. 1994, pp. 140–149. DOI: 10.1109/pdis.1994.331722.

[14] Nuno Preguiça. *Conflict-free Replicated Data Types: An Overview*. 2018. arXiv: 1806.10254.

[15] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. "Conflict-free replicated data types". In: *Proceedings of the 13th International Conference on Stabilization, Safety, and Security of Distributed Systems*. SSS'11. Grenoble, France: Springer-Verlag, 2011, pp. 386–400. ISBN: 9783642245497. DOI: 10.1007/978-3-642-24550-3_29.

[16] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA, Jan. 2011, p. 50. URL: https://inria.hal.science/inria-00555588.