Efficient Synchronization of CRDTs using VCube-PS

Leonardo de Freitas Galesky
Western Parana State University (UNIOESTE)
Cascavel, Paraná, Brazil
leonardo.galesky@unioeste.br

Luiz Antonio Rodrigues
Western Parana State University (UNIOESTE)
Cascavel, Paraná, Brazil
luiz.rodrigues@unioeste.br

Elias Procópio Duarte Jr.
Federal University of Parana (UFPR)
Curitiba, PR, Brazil
elias@inf.ufpr.br

Luciana Arantes
Sorbonne Universités, LIP6/CNRS
Paris, France
luciana.arantes@lip6.fr

ABSTRACT

This paper presents VCube-Sync, a system that uses a virtual hypercube topology as the basis for replication of a Conflict-free Replicated Data Types (CRDT) based data store. CRDT can ensure consistency in a deterministic and conflict-free manner. At the same time, hypercubes have been previously used for message distribution due to their fault tolerance and logarithmic latency, and also enable heuristics based on knowledge of the structured overlay. The protocol presented in this paper is based on VCube-PS, a publish-subscriber based on hypercube topology, and exploits the synergies between the latter and replication systems. Evaluation experiments with VCube-Sync in the context of operations-based CRDTs were conducted on the Grid5000 testbed under various loads and network distributions. The results were compared with those of EcoSyncTree, another replication protocol developed in recent research. The results show that VCube-Sync provides better performance in terms of latency, scalability, and bandwidth.

CCS CONCEPTS

• **Computer systems organization → Dependable and fault-tolerant systems and networks**; **Availability**; *Distributed architectures*; • **Information systems → Data structures**.

KEYWORDS

Conflict-Free Replication Data Types, vCube, replication, distributed algorithms

ACM Reference Format:
Leonardo de Freitas Galesky, Luiz Antonio Rodrigues, Elias Procópio Duarte Jr., and Luciana Arantes. 2023. Efficient Synchronization of CRDTs using VCube-PS. In *12th Latin-American Symposium on Dependable and Secure Computing (LADC 2023), October 16–20, 2023, La Paz, Bolivia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3569902.3569948

1 INTRODUCTION

Distributed systems provide large-scale Internet services, such as collaborative text editing [16], distributed databases, distributed version control systems, and real-time collaborative applications, such as online chat, and gambling [19]. In this context, models with weaker consistency guarantees, such as eventual consistency, enable lower latency and higher availability. In this setup, replicas can be independently modified and temporarily diverge from each other, relying on a deferred mechanism to broadcast and merge updates [23, 27].

However, since the order of operations cannot be determined globally, concurrent updates raise the possibility of conflicts that must be arbitrated by the participants in the system. Conflict-Free Replicated Data Types (CRDT) are data structures that satisfy mathematical specifications that ensure that operations can be performed independently and concurrently without any form of coordination while guaranteeing strong eventual consistency after synchronization, and tolerating network and replication delays [25]. These properties make them a powerful tool for building scalable and highly available distributed systems.

There are two original types of CRDT in terms of how they are synchronized: *State-based* and *Operation-based*. State-based CRDT must exchange all their local states to achieve consistency. This strategy is inefficient since the size of the state, and thus the payload, increases over the time [25]. On the other hand, CRDT based on operations propagates only a concise representation of the operation that changed the local state, resulting in much smaller messages and thus less bandwidth. However, this strategy requires that operations are transmitted over reliable channels that guarantee causal order delivery in most cases, adding overhead in the form of metadata and making the system less tolerant of packet loss and membership changes [21].

An emerging alternative is *Delta-State* CRDT, where only the differences between replicas are synchronized. The guarantees are similar to the state-based implementation but without the high transmission cost. In this optimized structure, it is usually necessary to introduce complementary algorithms to compute the deltas and reduce the entropy of the system [3].

Several software solutions implement CRDT to provide distributed data, such as databases like Riak [7] or distributed paradigm frameworks like Phoenix [17]. These solutions sometimes implement their own protocols to deal with dissemination, cluster node organization, and causal ordering of messages. Some works, such as [26] and [18], present specialized protocols for CRDT synchronization and have validated the benefits of tree-based solutions such as the Plumtree [15]. However, these implementations rely

LADC 2023, October 16–20, 2023, La Paz, Bolivia

ACM ISBN 978-1-4503-9737-7/22/11.
https://doi.org/10.1145/3569902.3569948

on unstructured network overlay topologies, limiting the membership inferences that could be leveraged to develop more efficient synchronization strategies [8, 11].

The VCube algorithm [10] defines a structured topology based on virtual hypercubes that have important logarithmic properties, such as the path length between two processes and the number of messages exchanged. The VCube-PS [9], a topic-based *publish-subscribe* system (*pub-sub*), exploits these properties to achieve efficient message dissemination, especially in hot topic scenarios, while guaranteeing causally ordered message delivery. It has not yet been explored as a strategy for distributing replica updates.

In this paper, we aim to lay the groundwork for a project that presents optimizations for CRDT synchronization in a massively distributed system using VCube-PS as the basis for the implementation of a replication protocol while also exploring the potential of partial replication over topics.

The remainder of this paper is organized as follows. Section 2 introduces the main types of CRDT, causal ordering, and replication protocols, including VCube-PS. Related work is summarized in Section 3. The methodology to complete the current proposal is defined in Section 4. Section 5 presents some evaluation results, comparing VCube-PS withe tha ECO-Sync-Tree, while Section 6 concludes the paper and presents some future directions.

2 BACKGROUND

This section introduces the CRDT main concepts and the replication protocols. It also includes VCube-PS, which is the core of VCube-Sync.

2.1 CRDT

Conflict-Free Replicated Data Types (CRDT) [3] are data types designed to be modified concurrently that have proven mathematical properties that ensure strong eventual consistency in a deterministic and conflict-free manner. The goal of CRDTs is to allow concurrent updates to replicas without the need for coordination or consensus algorithms, thereby enabling high availability and low-latency operations.

CRDTs are designed to work in scenarios where network partitions, message delays, or node failures can occur, leading to potential conflicts in data updates. They achieve this by ensuring that conflicts can be resolved automatically at each replica, guaranteeing that all replicas eventually converge to the same consistent state.

The fundamental classes of CRDT are *state-based*, *operation-based*, and *delta-state-based* [20]. In short, in the state-based model, the full local state is propagated to all replicas, and conflicts are resolved by merging the states. In the operation-based model, replicas exchange and merge individual operations (e.g., add, remove, increment) to achieve consistency. In our research, we will focus on operation-based CRDT.

2.1.1 State-Based CRDT. State-based CRDTs are also referred to as *convergent* replicated data types (CvRDTs), where the full state received from a replica must be merged by a commutative, associative, and idempotent function. CvRDTs achieve this by encoding the causality of operations, allowing replicas to merge their local updates in any order. Examples of CvRDTs include counters, sets, registers, and graphs [24].

A state-based CRDT often relies on a semilattice structure to represent the state. A semilattice is a partially ordered set where any two elements have a unique least upper bound (join) and greatest lower bound (meet). The merge function of the CRDT operates within this semilattice structure to ensure conflict resolution. More formally, a state-based CRDT can be defined by the tuple $(S, \sqsubseteq, \sqcup)$ where S is a *join-semilattice*, $\sqsubseteq$ represents partial order, and $\sqcup$ is a union operator (*join*) that derives the least upper bound (LUB) for all elements of S, such that $\forall s,t,u \in S$:

$$s \sqcup s = s \qquad \text{(idempotency)}$$
$$s \sqcup t = t \sqcup s \qquad \text{(commutativity)}$$
$$s \sqcup (t \sqcup u) = (s \sqcup t) \sqcup u \qquad \text{(associativity)}$$

The above properties guarantee that message reordering or duplication will not adversely affect the final convergence of the system. It is also tolerant of packet loss as long as a message containing this state or a higher-order message is received in the future. The main drawback of this implementation is the potentially unlimited state growth, especially considering that system synchronization is done by propagating the current local state of each node [25].

Shapiro et al. [24] present some examples of CRDT. One of them is a set of unique, non-removable elements E, also called a grow-only set (*GSet*). We can define it as a CRDT by taking the set of all subsets of E; $\sqsubseteq$ as the presence of the element in the set and $\sqcup$ as the union of two subsets. The subsets for $E = \{a, b, c\}$ can be represented by the Hasse diagram of Figure 1. Each element of E is represented as a vertex in the plane and a line goes upward from one vertex x to another vertex y whenever y covers x, i.e., if element x is less than or equal to element y, there is a line or directed edge connecting then. This representation allows for the visualization of the partial order and the relationships between the elements.

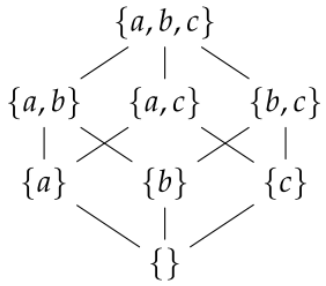

Figure 1: Hasse diagram for the $GSet\{a, b, c\}$

Another example of a state-based CRDT is the increment-only counter presented in Algorithm 1. The payload, in this case, is an array where each position corresponds to a node in the system. Each node increments by n the entry in the array that corresponds to itself and then broadcasts the updated state, e.g., the entire array, to other members.

2.1.2 Operation-Based CRDT. Operation-based (Op-based) CRDTs are more spatially efficient than state-based ones because they

Algorithm 1 State-based CRDT increment-only counter at proc. i [25]

```
function Init( )
    val = []

function Query( )
    return sum(val)

function Local_Update(n)
    val[i] = val[i] + n

function Merge(X, Y)
    for r ∈ X.val.keys ∪ Y.val.keys do
        val[r] = max(X.val[r], Y.val[r])
```

Algorithm 2 Op-based CRDT increment-only counter [25]

```
function Init( )
    val = 0

function Generator( )
    return ⟨inc, n⟩

function Effector(n)
    val = val + n
```

broadcast only a representation of the operations occurring locally at each node rather than the full state. Each operation represents a change to the state of the replicated data structure, such as adding or removing an element, or modifying a value. Operations are typically defined as functions that take the current state and additional parameters as input and produce a new state as output.

Op-based CRDTs are also called *commutative* replicated data types (CmRDTs) since the order of concurrent updates has no impact on the final converged state of the replicas. CmRDTs are typically implemented using conflict-free merge functions that combine concurrent updates deterministically. Examples of CmRDTs include state-based CRDTs like OR-Set (observed-remove set) and LWW-Element-Set (Last-Write-Wins Element Set).

For each update operation, Op-based CRDT must define two functions [21]:

Generator is executed in the replica from which the update is initiated. It has no side effects and generates an Effector that encodes the side effects of the operation.

Effector applies the side effect to all replicas and updates the state.

For most op-based CRDTs, a reliable channel and causal ordering of update operations are required. This requirement is usually achieved by middlewares, or other additional modules that implement protocols that do not require consensus and may be tolerant to partitioning, i.e., replicas connected in one partition of the network can communicate their operations with each other and eventually deliver the message to all other replicas [5, 21]. This is typically done by assigning unique identifiers (timestamps, sequence numbers, or other unique identifiers) to operations when they are generated. These identifiers enable the detection of concurrent operations and ensure that their effects can be correctly merged.

An operation-based counter is presented in Algorithm 2. The payload is a serialized representation of the *Effector* containing the command and an integer. The local initial value of the counter is zero.

2.1.3 Delta-State-Based CRDTs. Delta-state-based CRDTs (δ-CRDTs) incorporate the guarantees of state-based CRDTs but implement message size reduction techniques such as operations-based ones. Each replica tracks the changes made to its local copy of the data as a series of *deltas*, which are essentially the individual operations applied to the data structure. These deltas are typically represented in a compact and efficient form. When replicas communicate with each other, they exchange and merge these deltas to update their local state and achieve convergence. This is possible by implementing *delta-mutators* which return the *δ-state*, a representation of the effect of a sequence of updates on the state. This δ can be stored in a *buffer* which in time is transmitted to remote replicas [2].

One of the challenges of δ-state-based CRDT is the propagation criteria. For instance, for the sake of performance, it should avoid sending redundancy. Without this control, performance can be worse than in the fully state-based model. Some of the synchronization optimizations proposed by [11] use metadata related to the topology and membership of the network to achieve better results concerning used bandwidth, memory consumption, and processing time until convergence.

Despite these challenges, delta-based CRDTs have been successfully used in various applications. For example, collaborative text editing systems often employ delta-based CRDTs to track and merge changes made by multiple users in real-time, allowing them to work concurrently without conflicts.

2.2 Causal Ordering

Given a topic t and two messages m and m' and the partial order $\prec$ that represents a happen-before relationship, if $m \prec m'$ then the causal ordering of the delivery of a message guarantees that all replicas will only observe the effects of m' after they observe the effects of m.

VCube-PS uses causal barriers [4] to ensure the causal delivery of operations. The advantage of this model over logical vector clocks, for instance, is that it is not always necessary to transmit the whole clock entries to the system members. Message dependency is established directly which compresses the space used for metadata.

Consider two application-generated messages, m and m', published to a topic t. If the publication of m causally precedes the publication of m'; and there is no message m'' in which the publication of m causally precedes the publication of m'' while simultaneously m'' causally precedes the publication of m', then m is an immediate predecessor, or direct dependency, of m'. The set of messages that are direct dependencies of m determines the causal barrier (cb_m).

Since the causal barrier stores only direct dependencies, whenever a new message m is sent, the previous barrier is cleared, removing unnecessary metadata and reducing causal redundancy, since all previous messages would be indirect dependencies of m.

2.3 Replication Protocols

Data replication is often used as a fault-tolerant and performance-enhancing mechanism in distributed systems. Different replicas can maintain copies of the information, ensuring that there is no single point of failure and potentially spreading the load of accesses. In addition, it is possible to optimize system latency by keeping replicas closer to users. To ensure a consistent state even during runtime, these replicas must be able to communicate through a replication protocol [23].

In centralized replication systems, all communication between nodes is directed by a leader. In contrast, in decentralized replication systems, communication takes place between arbitrary nodes. This exchange of messages is done through primitives that implement broadcasting to all nodes or only to some (*multicast*). After a message is received, its delivery (*deliver*) can be intentionally delayed to satisfy a predefined condition such as the causal order [6].

When an operation occurs locally on a node, the change must be communicated to the other members of the system. The replication protocol defines when, how, and to whom such a change will be sent. In this case, nodes need to be connected to each other but the actual network topology can be overlaid by various optimized application-layer topologies [23].

In general, system members only need to be aware of their neighbors, which is defined by the overlay. In structured overlays, a relationship between nodes must be established in advance. This information and the properties of the network can therefore be leveraged by the application [8, 14, 23].

At the same time, synchronization can be more scalable and efficient if, instead of full replication, where all nodes keep a complete copy of the data, partial replication, where nodes can contain only a subset defined by a partitioning rule is used. This choice reduces the cost of propagating changes since only a subset of replicas needs to receive each update [13].

The replication protocol may also provide certain guarantees, such as the causal delivery of messages. Some common designs that provide such a causality are those based on vector clocks [5], which require additional storage as well as others based on causal barriers, which are more scalable, requiring negligible space, but can be less accurate in representing direct dependencies [10].

The Plumtree and VCube-PS message dissemination protocols are introduced and described below. Plumtree is the core of EcoSyncTree, whose performance is compared to the one of VCube-Sync in Section 5. It is worth noting that the Plumtree protocol is a system built on top of unstructured networks and specifies distribution rules through trees at runtime. The VCube-PS protocol is structured and therefore establishes node ordering rules over message dissemination time. Both provide causal delivery guarantees.

2.3.1 Plumtree. The *Plumtree* protocol [15] is unstructured and capable of optimizing message dissemination in gossip-based environments. Its implementation is completely decentralized and has fault-tolerance mechanisms. It is the basis for a series of implementations of replication systems that use CRDTs, such as those presented by [26] and [18].

The algorithm works by performing a configuration cycle that occurs only when necessary: after the exit or entry of a new node in the set. During the transmission of this message, the path traced by it is identified, that is, which were the network nodes involved in the dissemination, determining, from this, a tree that acts as an overlay of the initial topology. In this phase, the redundant paths are also identified [15].

Once the overlay is defined, the protocol makes use of two gossip strategies:

Eager push gossip Sends messages through the tree, built in the last configuration.

Lazy push gossip Uses redundant connections between nodes to send a small control message capable of validating receipt of the main message. If the target node does not receive the main message within a pre-established *timeout*, it can request it (*pull request*) through this alternative link. This *push-pull* mechanism introduces high availability and a way to repair the tree

2.3.2 VCube-PS. VCube is a distributed diagnosis algorithm that builds a hypercube-like overlay topology organizing the nodes of the system into progressively larger clusters. It is dynamically re-organized in case of a process failure and exhibits important logarithmic properties even in the presence of failures. A strategy to disseminate diagnostic information was also defined as part of the protocol and experimental results have shown that it presents latency improvements when compared to similar models [10].

In VCube, a node of index i groups all other $N-1$ members of the system into $d = \log_2 N$ similar sized clusters $1..d$. The list of nodes on each cluster s is defined by a function $c_{i,s}$ below where $\oplus$ is the logical bitwise exclusive operator XOR [10].

$$c_{i,s} = i \oplus 2^{2-1} || c_{i \oplus 2^{s-1},k} | k = 1, ..., s-1 \quad (1)$$

The graph on Figure 2 shows the hierarchical cluster-based logical organization of a VCube where $N = 8$ and $i = 0$.

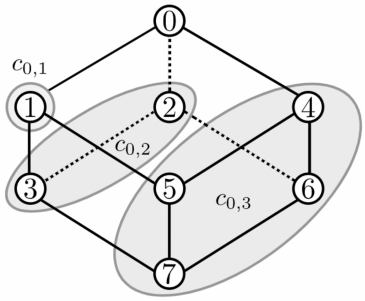

Figure 2: Virtual hypercube for node $i = 0$ [9]

The topic-based pub-sub system VCube-PS [9] exploits these properties to achieve efficient message dissemination, fault tolerance, and causal delivery of messages through the use of causal barriers. It is particularly effective in hot topic scenarios, which are common in large Internet systems.

Each topic arranges its members in a tree that is an *overlay* of the entire topology. Only the members of a topic receive its messages, and relays, if any, are temporary. In addition, each node (publisher) acts as the root or source of a message. In VCube-PS,

information about membership, as well as published messages, are sent to subscribers of a topic through dynamically constructed spanning trees whose root is at the publisher. Causal order per topic is ensured for message delivery.

A message is delivered by calling the VCube-PS function `Co_Deliver`. The delivery of it takes place after all its causal dependencies have been already delivered. Messages can be kept in a buffer until this condition is met.

VCube-PS has not yet been used as a replication strategy. It is worth pointing out that selectivity over topics could possibly be used as a criterion for partial replication.

Therefore, we propose in this article an experimental implementation and evaluation of a VCube-based replication protocol for CRDT-based data stores on top of VCube-PS.

The goal of our proposal is that, by transmitting update messages over the VCube-PS, which also allows inference of membership metadata, combined with guarantees such as causal delivery, the replication systems offer an efficient solution in terms of data freshness or latency, scalability, and fault tolerance.

3 RELATED WORK

Cure [1] is a decentralized replication protocol for key-value databases that provides causal consistency and atomicity, i.e., consistent concurrent operations on multiple keys, which is a form of high-availability transactions (HATs). To ensure that concurrent operations converge to the same correct value, the model uses operation-based CRDT and vector clocks to track causality. The latter are also used to control different versions of the same data set that can be garbage collected. This protocol is the foundation of the AntidoteDB database. One of the limitations of this system is that the metadata tracking mechanism grows with the number of nodes involved. The present work assumes massively distributed systems, which makes the implementation of Cure infeasible.

Selective Hearing [18] is a model that extends the existing LASP framework, which has CRDT as its primary data structure and has mechanisms for communication and synchronization between processes, with an epidemic dissemination structure based on the Plumtree protocol. The final product also enables a partial view of the cluster through overlays and has been presented as useful for applications in the Internet of Things and mobile gaming. This strategy aims to reduce latency for data visibility while maintaining dynamic membership. Since this implementation is based on the LASP programming model, it is not environment agnostic. Furthermore, unlike the present work, it considers an unstructured overlay.

SYNC Tree [26] is a broadcast protocol based on a modified version of the Plumtree and HyParView protocols. It provides causality guarantees and is capable of synchronizing hundreds of nodes. This model supports dynamic membership and introduces a synchronization strategy for joining and leaving nodes. Furthermore, the system uses CRDT as a distributed data structure and implements procedures such as garbage collection to achieve better latency and reduce communication costs. Leveraging all these mechanisms, extended ECO SYNC tree [26] is an optimized version of the SYNC protocol. Support for partial replication is suggested only as future work, and the network topology is unstructured.

The above discussed models have similar features to our proposal, but none of them delivers all of the properties introduced by this paper, i.e., (i) support for massively distributed systems using lightweight causality tracking; (ii) no central server; (iii) support to partial replication; (iv) usage of a structured topology that allows metadata to be inferred about nodes and their neighbors; and (v) modular implementation.

4 THE VCUBE-SYNC

In this section, we present VCube-Sync, a system where data are replicated N times without location restrictions and members are considered to be stable, i.e., VCube-Sync membership is not dynamic and members can not leave or join it. Through the use of reliable channels or message fair-lossy channels with message retransmission, VCube-Sync is considered to be tolerant to transient failures (e.g., network contention). It supports partial replication, allowing different nodes to have different data sets at the same time.

The application state is stored in the form of CRDT, enabling concurrent conflict-free operations. Synchronization is achieved by transmitting either a serialized representation of the operation, in the case of operation-based CRDT, or the complete state of a particular partition, in the case of state-based CRDT. The application is built in independent modules inspired by [26] and [12]:

membership determines the actual network connections between nodes;

broadcast applies diffusion strategies, such as overlays to the actual network, and determines diffusion trees while ensuring preconditions such as causality;

replication provides an interface to the CRDT data store;

application maintains the state of the data store using CRDT.

4.1 Proposed solution

The replication protocol of VCube-Sync exploits the VCube-PS topic-based publish-subscribe system (pub-sub) on top of VCube, described in Section 2.3.2.

Operations are only transmitted to interested nodes (subscribers) in a specific partition of data (topic). Each transmission occurs through VCube hierarchical hypercube-based diffusion tree, composed only of subscribers of that topic, with the root node being the one that originated the operation. VCube-PS protocol, via causal barriers, ensures that messages are delivered in causal order. It is also responsible for maintaining the mapping of nodes to topics. Thus, when node i initiates the transmission of message m, the latter is disseminated through a hierarchical transmission tree with i as the root, comprising nodes that are subscribers of a topic t.

In VCube-Sync a topic, the key for partial replication, maps one-to-one with a stored CRDT. A single node can be part of multiple topics and therefore stores multiple items. Messages can be of three types: `SUB` or "subscribe", which is sent by a node to express interest in a topic; `UNS` or "unsubscribe", which reverses the SUB operation; `PUB` or "publication", which in VCube-Sync corresponds to sending a replication message.

VCube-Sync exposes three functions: Subscribe(t), Unsubscribe(t), and Publish(t,m), where t represents a topic and m represents a message. A node can only publish messages in topics in which it participates. Once received, messages are delivered using the

function Co_Deliver, which happens only when all causal dependencies of the message are satisfied. As long as this condition is not met, the messages are kept in a buffer. The Algorithm 3 presents the VCube-Sync.

Algorithm 3 Application interface methods for process i [9].

```
1:  function Init( )
2:      counter ← 0
3:      view ← ∅

4:  function Subscribe(topic t)
5:      if ⟨i, SUB, _⟩ ∉ view[t] then
6:          view[t] ← {⟨i, SUB, counter⟩}
7:          Co_Broadcast(SUB, t, _)
8:          return OK
9:      return NOK

10: function Unsubscribe(topic t)
11:     if ⟨i, SUB, _⟩ ∈ view[t] then
12:         view[t] ← view[t] ∖ {⟨i, SUB, _⟩}
13:         Co_Broadcast(UNS, t, _)
14:         return OK
15:     return NOK

16: function Publish(topic t, message data)
17:     if ⟨i, SUB, _⟩ ∈ view[t] then
18:         Co_Broadcast(PUB, t, data)
19:         return OK
20:     return NOK
```

Two auxiliary functions Cluster(i, j) and Children(i, t, h) are also used to build the hierarchical diffusion trees, implemented according to [9]:

- Cluster(i, j): returns the index s of the cluster of a node i that contains node j, $(1 \leq s \leq \log_2 N)$. For example in Figure 2, Cluster$(0, 1) = 1$, Cluster$(0, 2)$ = Cluster$(0, 3) = 2$, and Cluster$(0, 4)$ = Cluster$(0, 5)$ = Cluster$(0, 6)$ = Cluster$(0, 7) = 3$.
- Children(i, t, h): returns a set containing all nodes virtually connected to i. A child of i is the first node of cluster $c_{i,s}$ which is also a subscriber of t, or simply the first node of this cluster without considering topics when this is not defined. The parameter h represents the cluster which can vary from 1 to $\log_2 N$. For $h = \log_2 N$, the function returns the set of first children of i; For any other value $h < \log_2 N$, the function returns only a subset of children, that is, only those where the number of cluster s is smaller than or equal to h $(s \leq h)$. For example, in Figure 2, if $t =$ ' $*$ ', Children$(0, *, 3) = \{1, 2, 4\}$, Children$(0, *, 2) = \{1.2\}$, and Children$(4, *, 2) = \{5.6\}$. On the other hand, if only nodes 0, 3, and 4 have subscribed to topic t_1, then Children$(0, t_1, 3) = \{3, 4\}$ and Children$(4, t_1, 2) = \emptyset$.

Algorithm 4 presents the message broadcasting mechanism for VCube-Sync synchronization. This algorithm is an adaptation of VCube-PS, having as differences (i) that VCube-PS also implements guaranteed FIFO (First In, First Out) reception *per-source* using messages of type ACK and (ii) that in VCube-PS, each node i stores which counter of the first message received from each node j. These mechanisms are implemented to handle node join and leave dynamics. In VCube-Sync we consider a stable system, so these mechanisms are not needed and have been removed which increases the responsiveness of the system by allowing more parallelism in transmission.

The broadcast of a replication message by the broadcast layer from a node i is initiated with the function Co_Broadcast(m), which creates the message to be broadcast. The function ReceiveMessage is then called and, from this point on, the behavior is the same whether for locally generated messages or for messages received over the network. This function starts the propagation of the message to the list of neighbors obtained by the function Children(i, t, h), and also the process of delivering the message locally. Local delivery of messages of type *SUB* corresponds to adding the message to the *recs* list and invoking the CheckDeliverable(t) function, while those of other types correspond to significant changes and therefore invoke the Update(set_1, set_2) method.

The CheckDelivery(t) method scans the *recs[t]* set which contains all messages received on topic t that has not yet been delivered. For each one, it is validated if the causal barrier is satisfied. In this case, the message is delivered by the Co_Deliver(m) method, removed o from the *recs[t]* set, and has its identifier added to the *delvs[t]* set and the causal barrier of this node i. Otherwise, the message remains in the *recs[t]* set where it will have the opportunity to be checked again when a new message from t is received.

Finally, in the context of VCube-Sync, the message data corresponds to a data replication operation. When a message is delivered by the Co_Deliver(m) method, the NotifyReply(m) method transmits the message to the *Replication* layer, where the operation is applied immediately based on the corresponding topic.

5 RESULTS

In order to evaluate the performance of VCube-Sync in different scenarios, experiments were carried out on the Grid'5000 platform (https://www.grid5000.fr) in Nancy region on the "gros" cluster by hosts equipped with an Intel Xeon Gold 5220 processor with 18 cores, 96 GB RAM and connected through a network with a capacity of 2x25 Gbps. Following Vieira [26] and Fouto et al. [12], Docker Swarm was used in the experiments, with one node of the replication protocol running in each container. The Babel framework [12], developed by the NOVA LINCS laboratory of NOVA University Lisbon, was also used for the simulation setup and communication between application layers and nodes.

VCube-Sync was compared with EcoSyncTree, which is based on unstructured topology with optimizations from the *Plumtree* and *HyParView* [26] protocols.

The experiments were conducted with different numbers of nodes: 50, 100, and 200, and different sets of publishers and subscribers, including scenarios of total and partial replication. These scenarios are grouped into two categories: those with only one publisher and those with multiple publishers. Each experiment was executed three times, and for experiments that require a node sample, the selection was made via uniform sampling. We used a CRDT based on register-type operations in which each operation has a

Algorithm 4 Causal broadcast of VCube-Sync at process i

```
1:  function Init( )
2:      ∀t ∈ TOPICS: view[t] ← ∅; recs[t] ← ∅; delv[t] ← ∅;
3:      counter ← 0; receivedIds ← ∅; causal_barrier[t] ← ∅

4:  function Co_Deliver(message m)
5:      NotifyReply(m)                          ▷ Deliver to the replication layer

6:  function Co_Broadcast(message_type type, topic t, content data)
7:      m = New(message)
8:      m.type ← type; m.s ← i; m.t ← t
9:      m.c ← counter; m.content ← data;
10:     m.cb ← causal_barrier[t]
11:     counter ← counter + 1
12:     causal_barrier[t] ← {⟨i, counter⟩}
13:     ReceiveMessage(m)

14: function ReceiveMessage(message m)
15:     if m.id ∈ receivedIds then
16:         return                              ▷ Ignore duplicated messages
17:     if m.type = SUB then
18:         chd ← Children(i, *, Cluster(i, m.s) − 1)
19:     else
20:         chd ← Children(i, m.t, Cluster(i, m.s) − 1)
21:     for all k ∈ chd do
22:         Send(m) to p_k
23:     if ⟨i, SUB, _ ⟩ ∈ view[m.t] then        ▷ i is a subscriber of m.t
24:         if m.type = PUB then
25:             recs[m.t] ← recs[m.t] ∪ {m}
26:             receivedIds ← msgs ∪ {m.id}
27:             CheckDeliverable(m.t)
28:         else
29:             view[m.t] ← Update(view[m.t], {⟨m.s, m.type, m.c⟩})


30: function Update(set_1, set_2)
31:     for all ⟨n_1, _, rc_1⟩ ∈ set_1 do
32:         if ∃ ⟨n_1, _, rc_2⟩ ∈ set_2 then
33:             if rc_2 > rc_1 then
34:                 set_1 ← set_1 ∖ {⟨n_1, _, rc_1⟩}
35:             else
36:                 set_2 ← set_2 ∖ {⟨n_1, _, rc_2⟩}
37:     return set_1 ∪ set_2

38: function CheckDeliverable(topic t)
39:     while (∃ m ∈ recs[t] : CheckCB(t, m.cb) = true) do
40:         Co_Deliver(m)
41:         recs[t] ← recs[t] \ {m}
42:         delvs[t] ← delvs[t] \ {⟨m.s, _⟩} ∪ {⟨m.s, m.c⟩}
43:         causal_barrier[t] ← causal_barrier[t] \ {m.cb}
                                ∪ {⟨m.s, m.c⟩}

44: function CheckCB(topic t, causal barrier cb)
45:     for all ⟨s, c⟩ ∈ cb do
46:         if ∃ ⟨s′, c′⟩ ∈ delvs[t] : s = s′ and c′ ≥ c then
47:             cb ← cb \ {⟨s, c⟩}
48:     return cb = ∅
```

size of 1,024 bytes. In all experiments, the number of nodes per machine was the same.

Average message delivery latency and bandwidth were both evaluated in various configurations with different number of nodes and topic arrangements. Application events are logged by logger Log4J. After generating logs in the distributed environment, they are analyzed by other *scripts*, which output the final results.

For each of the evaluated protocols, VCube-Sync and EcoSync-Tree, the configuration of the nodes was set according to Table 1.

Table 1: Configurations of the tests.

Type	Publishers	Subscribers
Single publisher	1	25%
Single publisher	1	100%
Multiple publishers	25% of members	100%
Multiple publishers	100% of members	100%

The execution of each experiment proceeds in the following sequence of steps: (1) Start all nodes through Docker containers; (2) Initiate the membership protocols and wait for stabilization; (3) Initiate the transmission of SUB-type messages, indicating interest in specific topics according to the arrangement of each experiment; (4) Initiate the replication message transmission during 400 seconds, with one message sent per second per publisher; (5) Upon completion of the transmission, the nodes remain available for up to 5 minutes to receive messages in transit or generated by nodes that initiated step 3 at a later time.

5.1 Single Publisher

This collection of experiments evaluates the replication performance in a scenario where only one node publishes messages, allowing the observation of the behavior without causal dependency. Figure 3(a) presents the average delivery latency when the percentage of nodes interested in the topic is 100% and 25%. The latter applies only to VCube-Sync since the ECO-Sync-Tree protocol does not provide partial replication. Thus, in this case, all nodes receive a replication message even if they are not interested in a particular topic.

The average latency for 200 nodes of ECO-Sync-Tree for total replication is 0.854s, a value 3.16 times higher than that observed in VCube-Sync in the same setup and 3.49 times higher than the scenario of partial replication. Moreover, the impact of partial replication can be observed in the total number of transmitted bytes for each simulation as depicted in Figure 3(b). For 200 nodes, the total number of bytes transmitted by the ECO-Sync-Tree was 142.67 megabytes, a value 4.68x greater than when using the VCube-Sync in the full replication arrangement, and 19.14x greater in the partial replication scenario. This suggests that the reduction factor induced by partial replication is as expected since only nodes interested in the topics receive and transmit messages, resulting in a linear reduction in the number of bytes proportionally.

It is also noteworthy that VCube does not require the sending of messages for topology maintenance. Therefore, byte transmission rate tends to zero if replication messages are not in transit. On the other hand, EcoSyncTree uses messages to maintain the topology, which makes the case of a single publisher not very optimized, since the maintenance cost corresponds to a portion of almost 50% of the

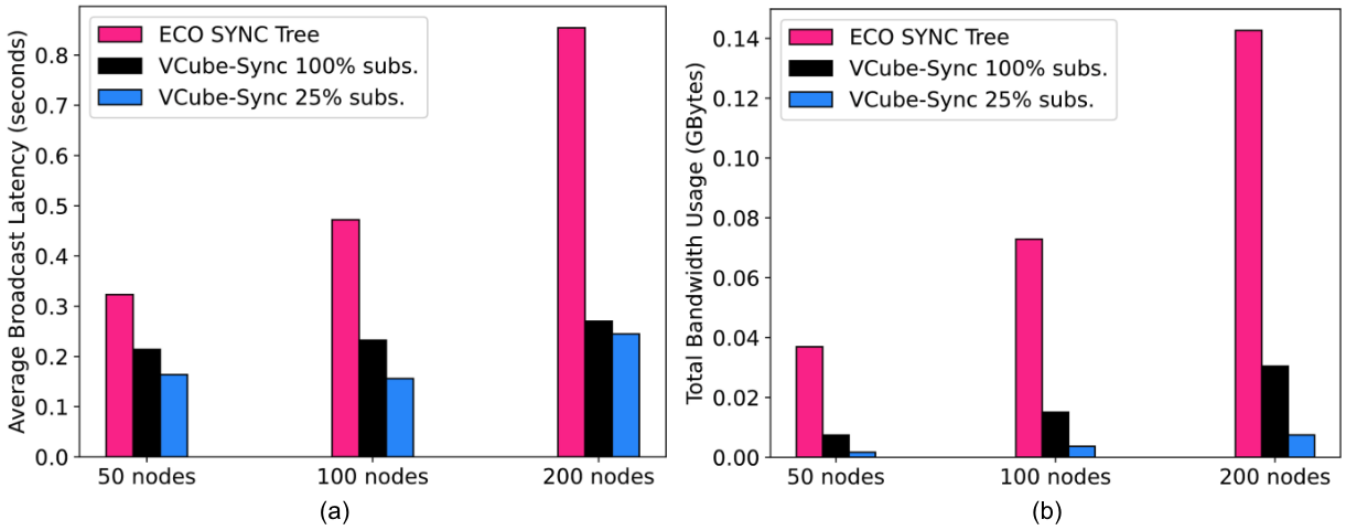

Figure 3: Results for a single Publisher. (a) Average message delivery latency in seconds per the protocol and the number of nodes; (b) Total experiment bandwidth usage in Gigabytes by protocol and the number of nodes.

network usage rate, as shown in Figure 4. Furthermore, the number of duplicate messages observed by nodes using ECOSyncTree with just one *publisher* ranged from 0 to 12%. Whereas for VCube-Sync there are never duplicate messages when there are no failures.

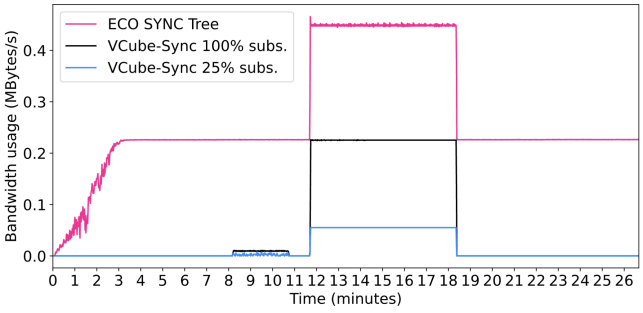

Figure 4: Bandwidth usage per protocol in megabytes per second for 200 nodes.

For both protocols, for all numbers of nodes in the 100% subscribers' scenario, the total number of messages transmitted over the network was the same. However, ECOSyncTree showed higher bandwidth usage compared to VCube-Sync. Sucha a diference can be explained since ECOSyncTree has a synchronization cycle that runs continuously even when no node has been identified as failing.

Figure 4 confirms that VCube-Sync algorithm performs significantly better than the Eco-Sync algorithm in terms of latency. Compared to Eco-Sync, VCube-Sync is capable of achieving much lower latencies across all node arrangements and subscriber percentages. It is also interesting to point out that as the number of nodes increases, latency of both algorithms increases as well. However, VCube-Sync is still able to maintain a relatively low latency compared to Eco-Sync, even with a higher number of nodes. Such a behavior suggests that VCube-Sync may be better suited for larger-scale systems that require more nodes.

5.2 Multiple Publishers

In this set of experiments, all nodes subscribe to a single topic, and the number of publishers is varied between 25% and 100% for each protocol. This setup allows us to evaluate the impact of message load balancing in each protocol. In the arrangement where only 25% of members are publishers, the selection is random and uniform.

Figure 5(a) shows that for 200 nodes, when there are 100% of publishers, the average message delivery latency of ECO-SYNC-tree is 1.56x that of VCube-Sync. When there exist only 25% of publishers, the difference is slightly reduced to 1.51x. This result demonstrates the efficiency of VCube-Sync in latency even in the presence of a large number of nodes, which is consistent with the results of Araujo et al. [9].

In Figure 5(b), VCube-Sync achieves a total throughput of 6.41 GB per second for the scenario with 200 nodes and 100% publishers, which is 7.39% higher than that of ECO-SYNC-tree. However, in the scenario with 25% publishers, VCube-Sync reduces the total bandwidth usage by 2.79%. The increase in the number of bytes in the 100% scenario is due to the additional metadata that a message may contain in VCube-Sync, especially those related to causal barrier.

As for bandwidth usage, according to Figure 5(b), for the scenario of 200 nodes and 100% of *publishers*, VCube-Sync presented a total throughput of 6, 41 GB, 7.39% more when compared to ECOSync-Tree. On the other hand, for the scenario with 25% of publishers, VCube-Sync had a 2.79% reduction in total bandwidth usage. The increase in the number of bytes in the 100% scenario is caused by the additional metadata that a message can contain in VCube-Sync, especially the causal barrier. Another observation is that, in a scenario with multiple publishers, the network maintenance overhead in ECOSyncTree is minimal, with the data rate for sending messages for network maintenance in the scenario with 200 nodes

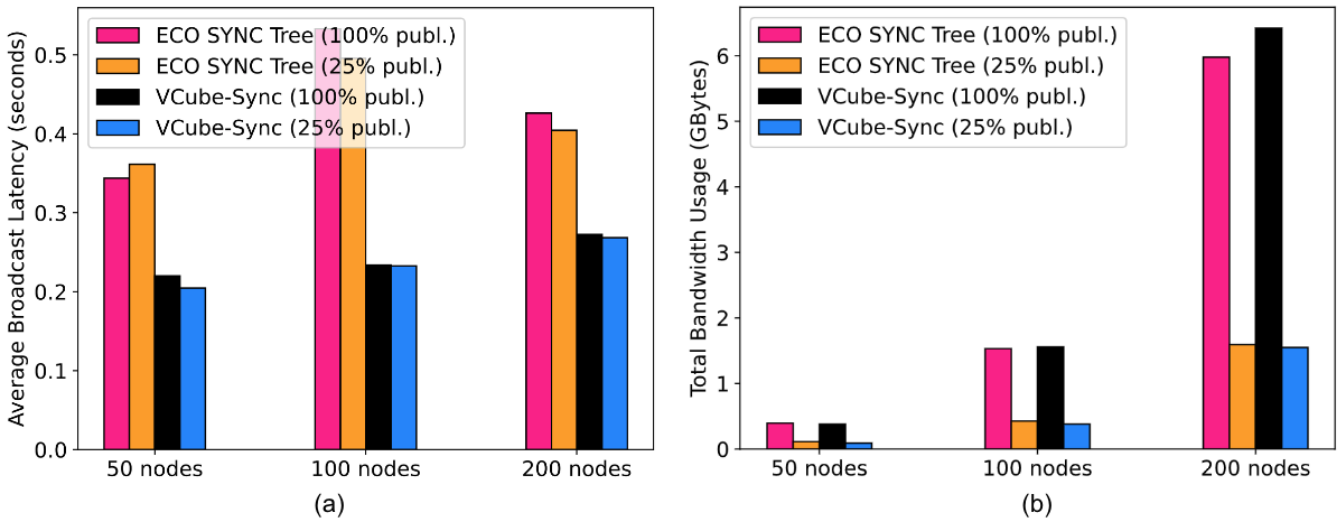

Figure 5: Result for Multiple Publishers. (a) Average message delivery latency in seconds per protocol, number of nodes and number of publishers; (b) Total experiment bandwidth usage in Gigabytes by the protocol, number of nodes and number of publishers.

and 100% of publishers being approximately 10 KBytes/s, while the maximum throughput of the experiment was 48.63 MBytes/s.

For both protocols, for every number of nodes in the 100% publishers' scenario, the total number of transmitted messages was 3% higher in VCube-Sync. A possible reason for such an increase is that, due to the intermediate synchronization performed by ECOSync-Tree, some of the replication messages are received after a synchronization that already has this event. Since redundant messages are not retransmitted, a small number of replication operations end up not traversing the entire tree and not counted as individual messages.

In Figure 5, it can be seen that VCube-Sync performs well in terms of latency when compared to Eco-Sync. The latency of VCube-Sync is consistently lower than that of Eco-Sync, especially in scenarios where all nodes are publishers. For example, in the case where there are 50 nodes and all of them publish messages, the latency of VCube-Sync is 220.10ms while that of Eco-Sync is 343.82ms. Likewise, in the case where there are 100 nodes and they all publish messages, the latency of VCube-Sync is 233.70ms while that of Eco-Sync is 532.93ms.

We can also observe that the percentage of publishers has a significant impact in latency of both algorithms. The greater the percentage of publishers, the higher the latency. For example, in the case where there are 50 nodes and only 25% of them publish messages, the latency of VCube-Sync is 204.66ms while that of Eco-Sync is 361.40ms. However, when all 50 nodes publish messages, the latency of VCube-Sync increases to 220.10ms, while that of Eco-Sync increases to 343.82ms.

The above results confirm that VCube-Sync provides significant advantages in message delivery latency with a small increase in the number of transmitted bytes.

6 CONCLUSION

In this paper, we introduce VCube-Sync, a new partial replication protocol based on hypercubes that offers causal ordering guarantees for operation delivery using causal barriers. The application state is maintained by CRDT, which allows operations to be performed concurrently while ensuring a deterministically convergent final state.

The proposed solution is based on the VCube-PS publish-subscribe protocol which is capable of creating hierarchical dissemination trees with the node that performed an operation updating the state as the root. Results show that the solution exhibits excellent performance in terms of message delivery latency and good usage rates in terms of bandwidth, especially in scenarios with multiple topics. These results suggest that the solution is a promising option for ensuring consistency in distributed data systems. Furthermore, VCube-PS, which serves as the foundation of VCube-Sync, performed well in simulated scenarios with more than a thousand nodes, ensuring the scalability of the solution.

As future work, we plan to add to VCube-Sync the capability to handle system membership dynamics and failure recovery. Although VCube can be used as a failure detector, the algorithm implemented in this work also requires additional state synchronization mechanisms similar to those implemented by ECO-Sync-Tree. Some existing solutions involve the hybrid use of types of CRDT, such as State or Delta-State-based CRDT, for this synchronization phase. Therefore, the latter could potentially use hypercube metadata to determine optimal synchronization methods.

In addition, to reduce the amount of messaging and communication costs, causality or time-based aggregation algorithms could be applied, such as those presented in [9] and [22].

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. Experiments have been carried out using the Grid'5000 testbed, which is supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and several universities and other organizations (https://www.grid5000.fr).

REFERENCES

[1] Deepthi Devaki Akkoorath, Alejandro Z. Tomsic, Manuel Bravo, Zhongmiao Li, Tyler Crain, Annette Bieniusa, Nuno Preguiça, and Marc Shapiro. 2016. Cure: Strong Semantics Meets High Availability and Low Latency. In *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 405–414. https://doi.org/10.1109/ICDCS.2016.98

[2] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2014. Efficient State-based CRDTs by Delta-Mutation. *CoRR* abs/1410.2803 (2014). arXiv:1410.2803 http://arxiv.org/abs/1410.2803

[3] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2018. Delta state replicated data types. *J. Parallel and Distrib. Comput.* 111 (jan 2018), 162–173. https://doi.org/10.1016/j.jpdc.2017.08.003 arXiv:1603.01529

[4] Roberto Baldoni, Ravi Prakash, Michel Raynal, and Mukesh Singhal. 1998. Efficient Δ-causal broadcasting. 13, 5 (Sept. 1998), 263–269.

[5] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. 2017. Pure Operation-Based Replicated Data Types. *CoRR* abs/1710.04469 (2017). arXiv:1710.04469 http://arxiv.org/abs/1710.04469

[6] Kenneth Birman, André Schiper, and Pat Stephenson. 1991. Lightweight Causal and Atomic Group Multicast. *ACM Trans. Comput. Syst.* 9, 3 (aug 1991), 272–314. https://doi.org/10.1145/128738.128742

[7] Russell Brown, Sean Cribbs, Christopher Meiklejohn, and Sam Elliott. 2014. Riak DT Map: A Composable, Convergent Replicated Dictionary. In *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency* (Amsterdam, The Netherlands) *(PaPEC '14)*. Association for Computing Machinery, New York, NY, USA, Article 1, 1 pages. https://doi.org/10.1145/2596631.2596633

[8] Miguel Castro, Manuel Costa, and Antony Rowstron. 2005. Debunking Some Myths about Structured and Unstructured Overlays. In *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*. USENIX Association, USA, 85–98.

[9] João Paulo de Araujo, Luciana Arantes, Elias P. Duarte, Luiz A. Rodrigues, and Pierre Sens. 2019. VCube-PS: A causal broadcast topic-based publish/subscribe system. *J. Parallel and Distrib. Comput.* 125 (2019), 18–30. https://doi.org/10.1016/j.jpdc.2018.10.011

[10] Elias P. Duarte, Luis C. E. Bona, and Vinicius K. Ruoso. 2014. VCube: A Provably Scalable Distributed Diagnosis Algorithm. *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, 17–22. https://doi.org/10.1109/ScalA.2014.14

[11] Vitor Enes, Paulo Sérgio Almeida, Carlos Baquero, and João Leitão. 2018. Efficient Synchronization of State-based CRDTs. *CoRR* abs/1803.02750 (2018). arXiv:1803.02750 http://arxiv.org/abs/1803.02750

[12] P. Fouto, P. Costa, N. Preguica, and J. Leitao. 2022. Babel: A Framework for Developing Performant and Dependable Distributed Protocols. , 146-155 pages. https://doi.org/10.1109/SRDS55811.2022.00022

[13] Pedro Fouto, João Leitão, and Nuno Preguiça. 2018. Practical and Fast Causal Consistent Partial Geo-Replication. In *IEEE 17th International Symposium on Network Computing and Applications (NCA)*. 1–10. https://doi.org/10.1109/NCA.2018.8548067

[14] J. Leitão. 2012. *Topology Management for Unstructured Overlay Networks*. Ph. D. Dissertation. Technical University of Lisbon.

[15] Joao Leitao, Jose Pereira, and Luis Rodrigues. 2007. Epidemic Broadcast Trees. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. IEEE, 301–310. https://doi.org/10.1109/SRDS.2007.27

[16] Geoffrey Litt, Sarah Lim, Martin Kleppmann, and Peter van Hardenberg. 2022. Peritext: A CRDT for Collaborative Rich Text Editing. In *Proceedings of the ACM on Human-Computer Interaction (PACMHCI)*. 35 pages. https://doi.org/10.1145/3555644

[17] Chris McCord. 2022. Phoenix Presence. https://github.com/phoenixframework/phoenix_pubsub/blob/master/lib/phoenix/tracker.ex

[18] Christopher Meiklejohn and Peter Van Roy. 2015. Selective Hearing: An Approach to Distributed, Eventually Consistent Edge Computation. In *2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW)*, Vol. 2016-Janua. IEEE, 62–67. https://doi.org/10.1109/SRDSW.2015.9

[19] Michael Owen. 2015. Using Erlang, Riak and the ORSWOT CRDT at bet365 for Scalability and Performance. http://www.erlang-factory.com/euc2015/michael-owen

[20] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. 2018. *Conflict-Free Replicated Data Types CRDTs*. Springer International Publishing, Cham, 1–10. https://doi.org/10.1007/978-3-319-63962-8_185-1

[21] Nuno M. Preguiça. 2018. Conflict-free Replicated Data Types: An Overview. *CoRR* abs/1806.10254 (2018). arXiv:1806.10254 http://arxiv.org/abs/1806.10254

[22] Luiz Rodrigues, Elias Duarte Jr, João Paulo de Araujo, Luciana Arantes, and Pierre Sens. 2018. Bundling Messages to Reduce the Cost of Tree-Based Broadcast Algorithms. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*. IEEE, 115–124. https://doi.org/10.1109/LADC.2018.00022

[23] Yasushi Saito and Marc Shapiro. 2005. Optimistic Replication. *ACM Comput. Surv.* 37, 1 (mar 2005), 42–81. https://doi.org/10.1145/1057977.1057980

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

[25] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 386–400.

[26] Ema Vieira. 2021. *ECO SYNC TREE: A Causal and Dynamic Broadcast Tree for Edge-based Replication*. Master's thesis. NOVA University Lisbon.

[27] Werner Vogels. 2009. Eventually Consistent. *Commun. ACM* 52, 1 (jan 2009), 40–44. https://doi.org/10.1145/1435417.1435432