Efficient Synchronization of State-based CRDTs

Article · March 2018

CITATIONS
5

READS
73

4 authors, including:

Vitor Enes
University of Minho
7 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Paulo Sérgio Almeida
University of Minho
53 PUBLICATIONS **919** CITATIONS
SEE PROFILE

Carlos Baquero
University of Minho
117 PUBLICATIONS **1,720** CITATIONS
SEE PROFILE

Efficient Synchronization of State-based CRDTs

Vitor Enes[1], Paulo Sérgio Almeida[1], Carlos Baquero[1], and João Leitão[2]

[1] HASLab/INESC TEC and Universidade do Minho
[2] NOVA LINCS, FCT and Universidade NOVA de Lisbon

Abstract. Data consistency often needs to be sacrificed in order to ensure high-availability in large scale distributed systems. *Conflict-free Replicated Data Types* (CRDTs) relax consistency by enabling query and update operations to be performed locally at any replica without synchronization. Consistency is achieved by background synchronization operations. In state-based CRDTs replicas synchronize by periodically sending their local state to other replicas and merging the received remote states into the local state. This can be extremely costly as the local state grows. Delta-based CRDTs address this problem by defining delta-mutators, which produce small incremental states (deltas) to be used in synchronization instead of the full state. However, current synchronization algorithms induce redundant wasteful delta propagation, namely in the general case of a network graph with alternative synchronization paths (desirable to achieve fault-tolerance). In this paper we explore this problem and identify two sources of inefficiency in current synchronization algorithms for delta-based CRDTs. We evolve the concept of join decomposition of a state-based CRDT and explain how it can be used to boost the efficiency of synchronization algorithms.

Keywords: Eventual Consistency, CRDTs, Join Decomposition

1 Introduction

Large-scale distributed systems often resort to replication techniques to achieve fault-tolerance and load distribution. These systems have to make a choice between availability and strong consistency, many times opting for the first [1,12]. A common approach is to allow replicas of some data type to temporarily diverge, making sure these replicas will eventually converge to the same state in a deterministic way. *Conflict-free Replicated Data Types* (CRDTs) [13,14] can be used to achieve this.

CRDTs come mainly in two flavors: *operation-based* and *state-based*. In both, queries and updates can be executed immediately at each replica, which ensures availability (as it never needs to coordinate beforehand with remote replicas to execute operations). In operation-based CRDTs [5,13], operations are disseminated assuming a reliable dissemination layer, that ensures exactly-once delivery of operations. State-based CRDTs need fewer guarantees from the communication channel: messages can be dropped, duplicated and reordered. When an update operation occurs, the local state is updated through a mutator, and from

time to time (since we can disseminate the state at a lower rate than the rate of the updates) the full (local) state is propagated to other replicas.

Although state-based CRDTs can be disseminated over unreliable communication channels [4], as the state grows, sending the full state becomes unacceptably costly. Delta-based CRDTs [2,3,15] address this issue, by defining delta-mutators that return a delta (δ), typically much smaller than the full state of the replica, to be merged with the local state. The same δ is also added to an outbound δ-buffer, to be periodically propagated to remote replicas. However, in the general case of a network graph with alternative synchronization paths, care must be taken in order to avoid redundant state being propagated between replicas. Delta-based CRDTs have been adopted in industry as part of Akka Distributed Data framework[3], and in other languages and systems.

In this paper we introduce the concept of join decomposition of a state-based CRDT and how it can be used to derive minimum delta-mutators and reduce the amount of state transmission necessary for delta-based synchronization with an improved synchronization algorithm. We experimentally evaluate the proposed solutions and show that they outperform classical approaches.

2 Synchronization of State-based CRDTs

A state-based CRDT can be defined as a triple $(\mathcal{S}, \sqsubseteq, \sqcup)$ where $\mathcal{S}$ is a join-semilattice (lattice for short, from now on), $\sqsubseteq$ its partial order and $\sqcup$ is a binary join operator that derives the least upper bound for any two elements of $\mathcal{S}$. State-based CRDTs are updated through a set of mutators $\mathcal{M}$ designed to be inflations, i.e., for every mutator $\mathsf{m} \in \mathcal{M}$, and every state $x \in \mathcal{S}$:

$$x \sqsubseteq \mathsf{m}(x)$$

Synchronization of replicas is achieved by having each replica periodically propagate its local state to other neighbour replicas. When a remote state is received, a replica updates its state to reflect the union of its local state and the received state. As the local state grows, more state needs to be sent, which might affect the usage of system resources (such as network) with a negative impact on the overall system performance. Ideally, each replica should only propagate the most recent modifications executed over its local state.

Delta-based CRDTs can be used to achieve this, by defining *delta-mutators* that return a smaller state which, when merged with the current state, generates the same result as applying the standard mutators, i.e., each mutator $\mathsf{m} \in \mathcal{M}$ has in delta-based CRDTs a corresponding δ-mutator $\mathsf{m}^\delta \in \mathcal{M}^\delta$ such that:

$$\mathsf{m}(x) = x \sqcup \mathsf{m}^\delta(x)$$

In this model, the deltas resulting from δ-mutators are added to a δ-buffer, in order to be propagated to neighbor replicas at the next synchronization step.

[3] https://doc.akka.io/docs/akka/2.4/scala/distributed-data.html

$$
\begin{aligned}
\mathsf{GCounter} &= \mathbb{I} \hookrightarrow \mathbb{N} \\
\bot &= \varnothing \\
\mathsf{inc}_i(p) &= p\{i \mapsto p(i) + 1\} \\
\mathsf{inc}_i^\delta(p) &= \{i \mapsto p(i) + 1\} \\
\mathsf{value}(p) &= \sum \{v \mid \langle k, v\rangle \in p\} \\
p \sqcup p' &= \{k \mapsto \mathsf{max}(p(k), p'(k)) \mid k \in l\} \\
&\textbf{where } l = \mathsf{dom}(p) \cup \mathsf{dom}(p')
\end{aligned}
$$

(a) Grow-only Counter

$$
\begin{aligned}
\mathsf{GSet}\langle E\rangle &= \mathcal{P}(E) \\
\bot &= \varnothing \\
\mathsf{add}(e, s) &= s \cup \{e\} \\
\mathsf{add}^\delta(e, s) &= \begin{cases} \{e\} & \textbf{if } e \notin s \\ \bot & \textbf{otherwise} \end{cases} \\
\mathsf{value}(s) &= s \\
s \sqcup s' &= s \cup s'
\end{aligned}
$$

(b) Grow-only Set

Fig. 1: Specifications of two data types, replica $i \in \mathbb{I}$

When a δ-group is received from a neighbor, it is also added to the buffer, for further propagation. This is required for causal consistency in systems that rely on partial views [9,11], i.e., systems where each replica is only aware of a small subset of the entire system's membership, a common practice to promote scalability [10].

2.1 CRDT examples

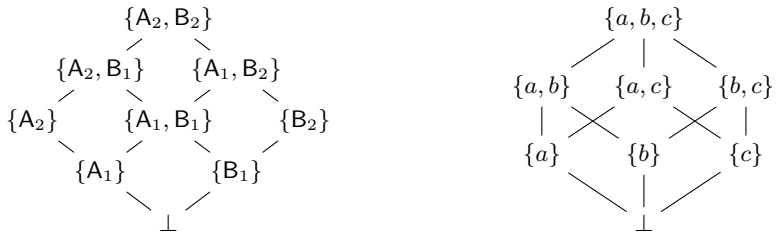
In Figure 1 we present the specification of two simple state-based CRDTs, defining their lattice states, mutators, corresponding δ-mutators, and the binary join operator $\sqcup$. These lattices are typically bounded and thus a bottom value $\bot$ is also defined. Note that the specifications do not define the partial order $\sqsubseteq$ since it can always be defined, for any lattice $\mathcal{S}$, in terms of $\sqcup$:

$$x \sqsubseteq y \Leftrightarrow x \sqcup y = y$$

A CRDT counter that only allows increments is known as *grow-only counter* (Figure 1a). In this data type, the set of replica identifiers $\mathbb{I}$ is mapped to the primitive lattice $\mathbb{N}$. Increments are tracked per replica, individually, and stored in a map entry. The value of the counter is the sum of each entry's value in the map. Mutator inc returns the updated map, while the δ-mutator inc^δ only returns the updated entry. The join of two $\mathsf{GCounter}$ computes, for each key, the maximum of the associated values.

The lattice state evolution (either by mutation or join of two states) can also be understood by looking at the corresponding Hasse diagram (Figure 2). For example, state $\{\mathsf{A}_1, \mathsf{B}_1\}$ in Figure 2a (where A_1 represents entry $\{\mathsf{A} \mapsto 1\}$ in the map, i.e. an increment by replica A), can result from an increment on $\{\mathsf{A}_1\}$ by B, from an increment on $\{\mathsf{B}_1\}$ by A, or from the join of these two states.

A *grow-only set*, Figures 1b and Figure 2b, is a set data type that only allows additions. Mutator add returns the updated set, while add^δ returns a singleton set with the added element, in the case where it was not in the set already.



(a) GCounter, with two replicas $\mathbb{I} = \{\mathsf{A}, \mathsf{B}\}$ (b) $\mathsf{GSet}\langle\{a, b, c\}\rangle$

Fig. 2: Hasse diagram of two data types

Although we have chosen here to illustrate the properties with very simple CRDTs, the results can be extended to more complex ones, such as add-wins sets and recursive maps. For further coverage of delta-CRDTs see [3].

2.2 Synchronization Cost Problem

In Figure 3 we illustrate a possible distributed execution of the classic delta-based synchronization algorithm [2,3], with three replicas $\mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{I}$ replicating a *grow-only set*. All replicas start with a bottom value $\bot = \varnothing$, each adding an element to the replicated set; synchronization with neighbors is represented by $\bullet$ and synchronization arrows are labeled with the state sent, where we underline or overline elements that are being redundantly sent and can be removed (thus improving network bandwidth consumption) by employing two novel optimizations that we introduce next.

At $\bullet^1$, B propagates the content of the δ-buffer (everything new since last synchronization step, thus $\{b\}$) to neighbours A and C. At $\bullet^2$, A sends all the content in the δ-buffer to B ($\{a\}$ from a local mutation, and the received $\{b\}$ from B), even though part of it came from B itself. By simply tracking the origin of each δ-group in the δ-buffer, replicas can **avoid back-propagation of δ-groups** (BP).

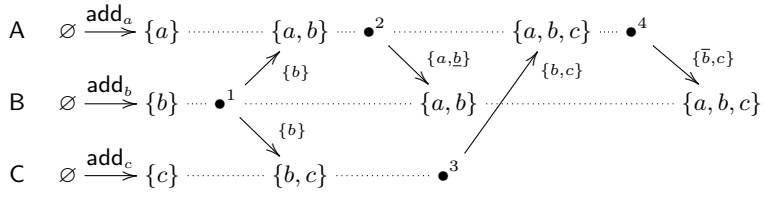


Fig. 3: Delta-based synchronization of a GSet with 3 replicas $\mathsf{A}, \mathsf{B}, \mathsf{C} \in \mathbb{I}$. Underlined and overlined set elements represent BP and RR optimizations, respectively.

At $\bullet^3$, the whole δ-buffer ($\{c\}$ from a local mutation, and $\{b\}$ received from B) has to be propagated from C to A. Upon receipt of δ-group $\{b, c\}$, A adds it to the δ-buffer and sends it to B at $\bullet^4$. However, part of this δ-group has already been in the δ-buffer (namely b), and thus, has already been propagated. This observation hints for another optimization: **remove redundant state in received δ-groups** (RR), before adding them to the δ-buffer.

3 Efficient Synchronization of State-based CRDTs

In this section we introduce state decomposition in state-based CRDTs, by exploiting the mathematical concept of *irredundant join decompositions* in lattices. We then demonstrate how this concept can be used to derive delta-mutators that are optimal, and obtain a more efficient delta-based synchronization algorithm.

3.1 Join Decomposition of a State-based CRDT

Definition 1 (Join-irreducible state). *State $x \in \mathcal{S}$ is join-irreducible if it cannot result from the join of any finite set of states $F \subseteq \mathcal{S}$ not containing x:*

$$x = \bigsqcup F \Rightarrow x \in F$$

Example 1. Let p_1, p_2 and p_3 be GCounter states, s_1, s_2 and s_3 be GSet states. States p_3 and s_3 are not join-irreducible states, since they can be decomposed into (i.e., result from the join of) two states different from themselves: $\{\mathsf{A}_5\}$ and $\{\mathsf{B}_7\}$ for p_3, $\{a\}$ and $\{b\}$ for s_3. Bottom (e.g., s_1) is never join-irreducible, as it is the join over an empty set $\bigsqcup \varnothing$.

✓ $p_1 = \{\mathsf{A}_5\}$	✗ $s_1 = \bot$
✓ $p_2 = \{\mathsf{B}_6\}$	✓ $s_2 = \{a\}$
✗ $p_3 = \{\mathsf{A}_5, \mathsf{B}_7\}$	✗ $s_3 = \{a, b\}$

In a Hasse diagram of a finite lattice (e.g., in Figure 2) the join-irreducibles are those elements with exactly one link below. Given lattice $\mathcal{S}$, we use $\mathcal{J}(\mathcal{S})$ for the set of all join-irreducible elements of $\mathcal{S}$.

Definition 2 (Join Decomposition). *Given a lattice state $x \in \mathcal{S}$, a set of join-irreducibles D is a join decomposition [6] of x if its join produces x:*

$$D \subseteq \mathcal{J}(\mathcal{S}) \wedge \bigsqcup D = x$$

In lattices satisfying the *descending chain condition* (DCC), see e.g., [7], which is usual in CRDTs, every element has a finite join decomposition.

Definition 3 (Irredundant Join Decomposition). *A join decomposition D is irredundant if no element in it is redundant:*

$$D' \subset D \Rightarrow \bigsqcup D' \sqsubset \bigsqcup D$$

Example 2. Let $p = \{\mathsf{A}_5, \mathsf{B}_7\}$ be a GCounter state, $s = \{a, b, c\}$ a GSet state, and consider the following sets of states as tentative decompositions of p and s.

✗	$P_1 = \{\{\mathsf{A}_5\}, \{\mathsf{B}_6\}\}$	✗	$S_1 = \{\{b\}, \{c\}\}$
✗	$P_2 = \{\{\mathsf{A}_5\}, \{\mathsf{B}_6\}, \{\mathsf{B}_7\}\}$	✗	$S_2 = \{\{a, b\}, \{b\}, \{c\}\}$
✗	$P_3 = \{\{\mathsf{A}_5, \mathsf{B}_6\}, \{\mathsf{B}_7\}\}$	✗	$S_3 = \{\{a, b\}, \{c\}\}$
✓	$P_4 = \{\{\mathsf{A}_5\}, \{\mathsf{B}_7\}\}$	✓	$S_4 = \{\{a\}, \{b\}, \{c\}\}$

Only P_4 and S_4 are irredundant join decompositions of p and s. P_1 and S_1 are not decompositions since their join does not result in p and s, respectively; P_2 and S_2 are decompositions but contain redundant elements, $\{\mathsf{B}_6\}$ and $\{b\}$, respectively; P_3 and S_3 do not have redundancy, but contain reducible elements (S_2 fails to be an irredundant join decomposition for the same reason, since its element $\{a, b\}$ is also reducible).

For typical CRDTs, not only is the state a join-semilattice, but it is also a distributive lattice satisfying DCC; therefore, (as corollary of the dual of Theorem 6 of [6]) every state x has a unique irredundant join decomposition. It is precisely in these that we are interested in to decompose CRDT states. Let $\Downarrow x$ denote this unique decomposition. From the Birkhoff's Representation Theorem[4], it is given by the maximals of the join-irreducibles below x:

$$\Downarrow x = \mathsf{max}\{r \in \mathcal{J}(\mathcal{S}) \mid r \sqsubseteq x\}$$

As two examples, for the GCounter and GSet data types, their (quite trivial) irredundant join decompositions of a counter state p and a set s are given by:

$$\Downarrow p = \{\{k \mapsto v\} \mid \langle k, v\rangle \in p\} \qquad\qquad \Downarrow s = \{\{e\} \mid e \in s\}$$

Having a unique irredundant join decomposition, we can define a function which gives the minimum delta, or "difference" in analogy to set difference, between two states:

$$\Delta(x, y) = \bigsqcup\{r \in \Downarrow x \mid r \not\sqsubseteq y\}$$

which when joined with y gives $x \sqcup y$, i.e., $\Delta(x, y) \sqcup y = x \sqcup y$. It is minimum in the sense that it is smaller than any other z which produces the same result:

$$z \sqcup y = x \sqcup y \Rightarrow \Delta(x, y) \sqsubseteq z$$

3.2 Minimum δ-mutators

If not carefully designed, δ-mutators can be a source of redundancy when the resulting δ-state contains information that has already been incorporated in the lattice state. As an example, the original δ-mutator add^δ of GSet presented in [2] always returns a singleton set with the element to be added, even if the element is already in the set (in Figure 1b we have presented a definition of add^δ that is optimal). By resorting to function Δ, minimum delta-mutators can be trivially derived from a given mutator:

$$\mathsf{m}^\delta(x) = \Delta(\mathsf{m}(x), x)$$

[4] Although stated for finite lattices, it can be applied to decompose a CRDT state s when the sublattice induced by the ideal $\downarrow s$ is finite, as is usually the case.

3.3 Delta-based Synchronization Revisited

Classic delta-based synchronization [2,3] is described formally in Algorithm 1. Each replica i maintains a lattice state $x_i \in \mathcal{S}$ (**line** 4) and a δ-buffer $B_i \in \mathcal{P}(\mathcal{S})$ as a set of lattice states (**line** 5). When an update operation occurs, the resulting δ is merged with the local state x_i (**line** 19) and added to the buffer B_i (**line** 20), resorting to function store. Periodically, the whole content of the δ-buffer B_i (**line** 11) is propagated to neighbors (**line** 12).

For simplicity, it is assumed that communication channels between replicas cannot drop messages (reordering and duplication is considered), and that is why the buffer is cleared after each synchronization step (**line** 13). This assumption can be removed by simply tagging each entry in the δ-buffer with a unique sequence number, and by exchanging acks between replicas: once an entry has been acknowledged by every neighbour, it is removed from the δ-buffer, as in [2].

When a δ-group is received (**line** 14), if it will induce an inflation in the local state (**line** 16), it is merged and added to the buffer, resorting to the same function store. Few changes are required in order to incorporate BP and RR optimizations in the classic algorithm, as shown in Algorithm 2.

Avoid back-propagation of δ-groups For BP, each entry in the δ-buffer is tagged with its origin (**line** 5 and **line** 20), and at each synchronization step with neighbour j, entries tagged with j are filtered out (**line** 11).

```
1  inputs:
2    n_i ∈ P(I), set of neighbors
3  state:
4    x_i ∈ S, state, x_i^0 = ⊥
5    B_i ∈ P(S), buffer, B_i^0 = ∅
6  on operation_i(m^δ)
7    δ = m^δ(x_i)
8    store(δ, i)
9  periodically // synchronize
10   for j ∈ n_i
11     d = ⨆ B_i
12     send_{i,j}(delta, d)
13   B'_i = ∅
14 on receive_{j,i}(delta, d)
15
16   if d ⋢ x_i
17     store(d, j)
18 fun store(s, o)
19   x'_i = x_i ⊔ s
20   B'_i = B_i ∪ {s}
```

Algorithm 1: Classic delta-based synchronization algorithm, replica $i \in \mathbb{I}$

```
1  inputs:
2    n_i ∈ P(I), set of neighbors
3  state:
4    x_i ∈ S, state, x_i^0 = ⊥
5    B_i ∈ P(S × I), buffer, B_i^0 = ∅
6  on operation_i(m^δ)
7    δ = m^δ(x_i)
8    store(δ, i)
9  periodically // synchronize
10   for j ∈ n_i
11     d = ⨆{s | ⟨s, o⟩ ∈ B_i ∧ o ≠ j}
12     send_{i,j}(delta, d)
13   B'_i = ∅
14 on receive_{j,i}(delta, d)
15   s = Δ(d, x_i)
16   if ⊥ ≠ s
17     store(s, j)
18 fun store(s, o)
19   x'_i = x_i ⊔ s
20   B'_i = B_i ∪ {⟨s, o⟩}
```

Algorithm 2: Delta-based synchronization algorithm, with BP and RR optimizations, replica $i \in \mathbb{I}$

Remove redundant state in received δ-groups A received δ-group can contain redundant state, i.e., state that has already been propagated to neighbors, or state that is in the δ-buffer B_i, still to be propagated. This occurs in topologies where the underlying graph is cyclic: nodes can receive the same information from different paths in the graph. In order to detect if a δ-group has redundant state, nodes do not need to keep everything in the δ-buffer or even inspect the δ-buffer: it is enough to compare the received δ-group with the local lattice state x_i. In Algorithm 1, received δ-groups were added to δ-buffer only if they would strictly inflate the local state (**line** 16). For RR in Algorithm 2, we extract from the δ-group what strictly inflates the local state x_i (**line** 15), and store it if it is different from bottom (**line** 16). This extraction is achieved by selecting which irreducible states from the join decomposition of the received δ-group strictly inflate the local state, resorting to function Δ.

4 Evaluation

In this evaluation we compare classic delta-based synchronization against state-based, and show the benefits of employing BP and RR optimizations under a range of workloads and different underlying network topologies.

Experimental Setup The evaluation takes place in a Kubernetes cluster with 16 Quad Core Intel Xeon 2.4 GHz, deployed in Emulab [16]. Figure 4 depicts the two network topologies employed in these experiments: a partial-mesh with 16 nodes, each node with 4 neighbors; and a tree with 14 nodes, each node with 3 neighbors, with the exception of leaf nodes.

We have designed a set of of micro-benchmarks, in which each node periodically (every second) synchronizes with neighbors and executes an update operation over a CRDT. The update operation depends on the CRDT type: GSet: an addition of a globally unique element to the set; GCounter: an increment on the counter; and GMap K%: each node updates some keys, such that globally K% of all the keys in the *grow-only map* are modified within each synchronization interval (we set the total number of keys to 1000).

The top of Figure 5 shows, for GSet and GCounter, the average transmission per node throughout the experiment, while the bottom shows the transmission

Fig. 4: Network topologies employed: a 16-node partial-mesh (to the left) and a 14-node tree (to the right).

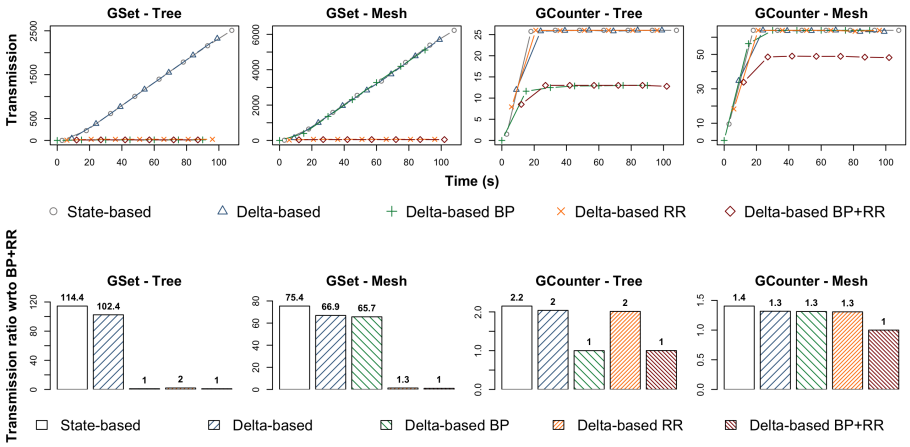


Fig. 5: Transmission of GSet and GCounter – tree and mesh topologies

ratio with relation to the best algorithm (delta-based with BP and RR). Transmission is measured by counting the number of elements in the GSet or the number of map entries in the GCounter. The first observation is that, in all four configurations, classic delta-based represents almost no improvement, when compared to state-based. In the tree topology, BP is enough to attain the best result, because the underlying topology does not have cycles, and thus, redundant state is not induced by the algorithm. With a partial-mesh, BP has little effect, and RR contributes most to the overall improvement.

Even with the optimizations proposed, the best result for GCounter is not much better than state-based. This is expected since most entries of the underlying map are being updated between each synchronization step: each node has almost always something new from every other node in the system to propagate (thus being similar to state-based in some cases). This pattern represents a special case of a map in which 100% of its keys are updated between state synchronizations. In Figure 6 we study other update patterns, by measuring the transmission of GMap 10%, 30%, 60%, and 100%. These results are further evidence of what we have observed in the case of GSet: BP suffices if the network graph is acyclic, but RR is effectively needed in the more general case.

The size of δ-groups being propagated, not only affects the network bandwidth consumption, but also the memory required to store them in the δ-buffer for further propagation. During the experiments we periodically measure the amount of state (both CRDT state and metadata required for delta-based synchronization) being stored in memory. In Figure 7 we present the average memory ratio with respect to state-based (the optimal case since only the CRDT state is stored). With the proposed optimizations, we improve previous approaches by up-to 270% less memory (ignoring GSet, since a longer run would result in a higher percentage), in some cases having almost no overhead when comparing to the optimal case.

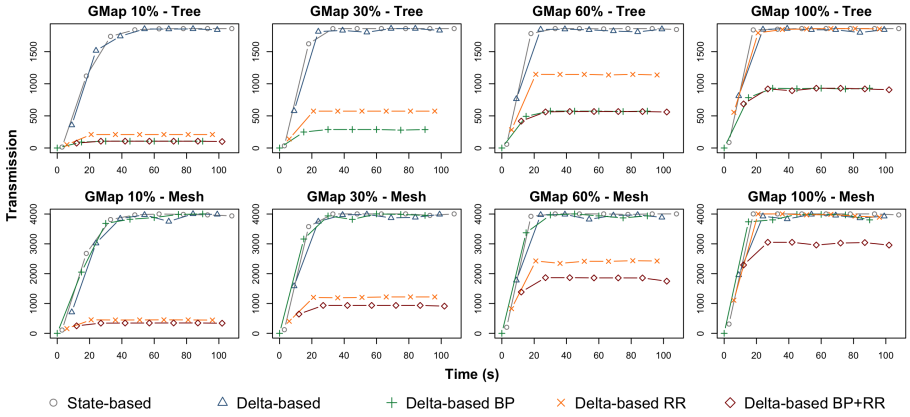


Fig. 6: Transmission of GMap 10%, 30%, 60% and 100% – tree and mesh topologies

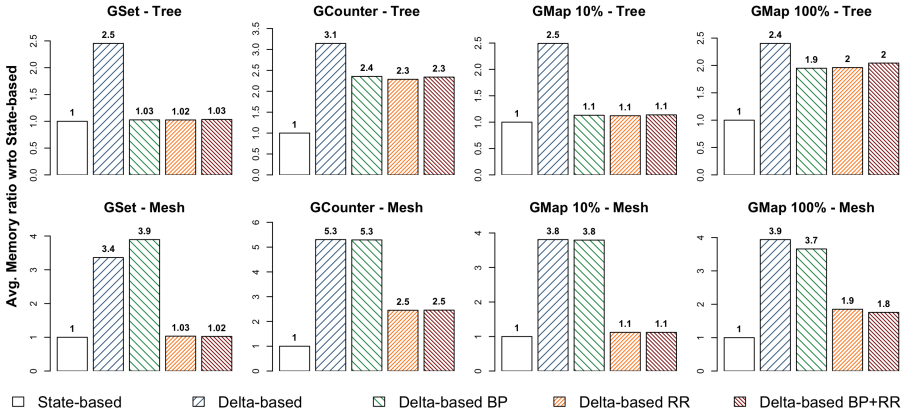


Fig. 7: Average memory ratio with respect to State-based (optimal) for GCounter, GSet, GMap 10% and 100% – tree and mesh topologies

Figure 8 reports the cumulative distribution for the total processing time of nodes when synchronizing the GMap with updates over 10% and 100% of the keys. We individually consider nodes sending updates and receiving updates.

There are two key observations. The first, since our solution sends less (redundant) information, it has consistently lower processing overhead than delta-based synchronization when sending. The second is that the processing overhead when receiving in a 100% workload is higher for our solution when compared with both delta and state-based alternatives. This is explained by the fact that RR tries to remove redundant state when there is almost none, representing a substantial overhead. We note however that this is an extreme workload that we do not expect to be a common case.

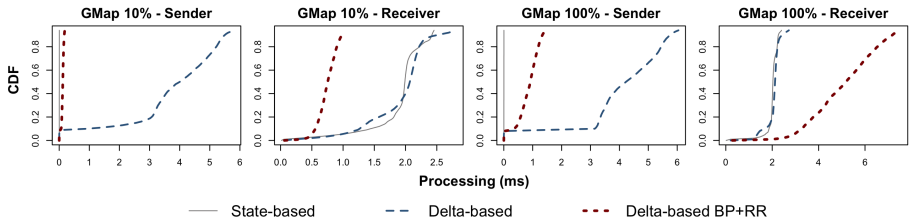
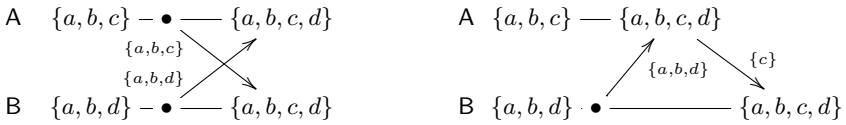


Fig. 8: CDF time producing (Sender) and processing (Receiver) messages – mesh topology

5 Further Applications of Join Decompositions

In classic delta-based synchronization each replica needs to keep track of which δ-group in the δ-buffer has been effectively received by its neighbors. When a δ-group is acknowledged by all neighbors, it is removed from the buffer. If a neighbor stops acknowledging (e.g., due to a network partition), the buffer will grow indefinitely, which might force garbage-collection of its content. Hence, the metadata required for delta-based synchronization is not available (this also occurs in systems with highly dynamic overlays, where the set of neighbors is constantly changing). One naive solution is to perform bidirectional full state transmission, as illustrated in Figure 9a. This is the strategy employed by classic delta-based synchronization [2,3].

State-driven synchronization The same technique used to remove redundant state in the received δ-groups and to derive minimum δ-mutators can also be used to design an alternative solution that decomposes the local state into smaller states, that are selected and grouped for efficient transmission in a pair-wise synchronization.



(a) Naive bidirectional full-state synchronization (b) *State-driven* synchronization

Fig. 9: Synchronization of a *grow-only set* with two replicas $\mathsf{A}, \mathsf{B} \in \mathbb{I}$.

As depicted in Figure 9b, in *state-driven* synchronization, B starts by sending its local state to A, and given this state, A is able to compute a state ($\{c\}$) that reflects the updates missed by B. This technique was initially presented in [8], a work in progress report, where we also devise a *digest-driven* approach.

References

1. P. Ajoux, N. Bronson, S. Kumar, W. Lloyd, and K. Veeraraghavan. Challenges to Adopting Stronger Consistency at Scale. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 13–13, Berkeley, CA, USA, 2015. USENIX Association.
2. P. S. Almeida, A. Shoker, and C. Baquero. Efficient State-Based CRDTs by Delta-Mutation. In *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers*, pages 62–76, 2015.
3. P. S. Almeida, A. Shoker, and C. Baquero. Delta State Replicated Data Types. *J. Parallel Distrib. Comput.*, 111:162–173, 2018.
4. P. Bailis and K. Kingsbury. The Network is Reliable. *Commun. ACM*, 57(9):48–55, Sept. 2014.
5. C. Baquero, P. S. Almeida, and A. Shoker. Pure Operation-Based Replicated Data Types. *CoRR*, abs/1710.04469, 2017.
6. G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
7. B. A. Davey and H. A. Priestley. Introduction to Lattices and Order. 1990.
8. V. Enes, C. Baquero, P. S. Almeida, and A. Shoker. Join Decompositions for Efficient Synchronization of CRDTs after a Network Partition: Work in progress report. In *First Workshop on Programming Models and Languages for Distributed Computing, PMLDC@ECOOP 2016, Rome, Italy, July 17*, page 6, 2016.
9. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, Nov. 2003.
10. J. Leitão. *Topology Management for Unstructured Overlay Networks*. PhD thesis, Technical University of Lisbon, Sept. 2012.
11. J. Leitão, J. Pereira, and L. E. T. Rodrigues. HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh, UK, Proceedings*, pages 419–429, 2007.
12. H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd. Existential Consistency: Measuring and Understanding Consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 295–310, New York, NY, USA, 2015. ACM.
13. M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, pages 386–400, 2011.
14. M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Convergent and Commutative Replicated Data Types. *Bulletin of the EATCS*, 104:67–88, 2011.
15. A. van der Linde, J. Leitão, and N. Preguiça. Δ-CRDTs: Making Δ-CRDTs Delta-based. In *Proceedings of the 2Nd Workshop on the Principles and Practice of Consistency for Distributed Data*, PaPoC '16, pages 12:1–12:4, New York, NY, USA, 2016. ACM.
16. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.

References

1. P. Ajoux, N. Bronson, S. Kumar, W. Lloyd, and K. Veeraraghavan. Challenges to Adopting Stronger Consistency at Scale. In *Proceedings of the 15th USENIX Conference on Hot Topics in Operating Systems*, HOTOS'15, pages 13–13, Berkeley, CA, USA, 2015. USENIX Association.
2. P. S. Almeida, A. Shoker, and C. Baquero. Efficient State-Based CRDTs by Delta-Mutation. In *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers*, pages 62–76, 2015.
3. P. S. Almeida, A. Shoker, and C. Baquero. Delta State Replicated Data Types. *J. Parallel Distrib. Comput.*, 111:162–173, 2018.
4. P. Bailis and K. Kingsbury. The Network is Reliable. *Commun. ACM*, 57(9):48–55, Sept. 2014.
5. C. Baquero, P. S. Almeida, and A. Shoker. Pure Operation-Based Replicated Data Types. *CoRR*, abs/1710.04469, 2017.
6. G. Birkhoff. Rings of sets. *Duke Mathematical Journal*, 3(3):443–454, 1937.
7. B. A. Davey and H. A. Priestley. Introduction to Lattices and Order. 1990.
8. V. Enes, C. Baquero, P. S. Almeida, and A. Shoker. Join Decompositions for Efficient Synchronization of CRDTs after a Network Partition: Work in progress report. In *First Workshop on Programming Models and Languages for Distributed Computing, PMLDC@ECOOP 2016, Rome, Italy, July 17*, page 6, 2016.
9. P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec. Lightweight Probabilistic Broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, Nov. 2003.
10. J. Leitão. *Topology Management for Unstructured Overlay Networks*. PhD thesis, Technical University of Lisbon, Sept. 2012.
11. J. Leitão, J. Pereira, and L. E. T. Rodrigues. HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast. In *The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2007, 25-28 June 2007, Edinburgh, UK, Proceedings*, pages 419–429, 2007.
12. H. Lu, K. Veeraraghavan, P. Ajoux, J. Hunt, Y. J. Song, W. Tobagus, S. Kumar, and W. Lloyd. Existential Consistency: Measuring and Understanding Consistency at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, pages 295–310, New York, NY, USA, 2015. ACM.
13. M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, pages 386–400, 2011.
14. M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Convergent and Commutative Replicated Data Types. *Bulletin of the EATCS*, 104:67–88, 2011.
15. A. van der Linde, J. Leitão, and N. Preguiça. Δ-CRDTs: Making Δ-CRDTs Delta-based. In *Proceedings of the 2Nd Workshop on the Principles and Practice of Consistency for Distributed Data*, PaPoC '16, pages 12:1–12:4, New York, NY, USA, 2016. ACM.
16. B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, Dec. 2002. USENIX Association.