Δ-CRDTs: Making δ-CRDTs Delta-Based

Albert van der Linde
a.linde@campus.fct.unl.pt

João Leitão
jc.leitao@fct.unl.pt
NOVA LINCS, DI, FCT,
Universidade NOVA de Lisboa

Nuno Preguiça
nuno.preguica@fct.unl.pt

ABSTRACT

Replication is a key technique for providing both fault tolerance and availability in distributed systems. However, managing replicated state, and ensuring that these replicas remain consistent, is a non trivial task, in particular in scenarios where replicas can reside on the client-side, as clients might have unreliable communication channels and hence, exhibit highly dynamic communication patterns. One way to simplify this task is to resort to CRDTs, which are data types that enable replication and operation over replicas with no coordination, ensuring eventual state convergence when these replicas are synchronized. However, when the communication patters, and therefore synchronization patterns, are highly dynamic, existing designs of CRDTs might incur in excessive communication overhead. To address those scenarios, in this paper we propose a new design for CRDTs which we call Δ-CRDT, and experimentally show that under dynamic communication patters, this novel design achieves better network utilization than existing alternatives.

CCS Concepts

•**Computer systems organization** → *Distributed architectures;*

Keywords

CRDT, Replication, Weak Consistency

1. INTRODUCTION AND CONTEXT

Web applications running in cloud infrastructures often use geo-replication for providing high availability and low latency to clients. To be able to continue operating during network partitions, these systems often adopt weakly consistent data replication protocols [2]. Such protocols allow replicas to be modified concurrently, requiring some reconciliation mechanism to merge these concurrent updates.

PaPOC'16, April 18-21 2016, London, United Kingdom
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4296-4/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2911151.2911163

CRDTs [5] have been proposed as a principled approach for providing convergence of general purpose data types. CRDTs come in two main flavors. State-based CRDTs synchronize by having replicas exchange their full local state (including metadata). This is inefficient when the size of these data objects grow significantly (for instance, in a large Set, the full Set needs to be propagated whenever a single element is added). The second flavor of CRDTs is called Operation-based CRDTs, where instead of exchanging the full state, replicas only propagate among them the operations that mutate their state. In this case, operations have to be propagated respecting the causality of operations, which not only introduces additional overhead (to keep track of causality) but also fits poorly in scenarios where there are large number of replicas, and where communication patterns among these replicas are highly dynamic, for instance, due to poor connectivity among these replicas.

A recent alternative, named δ-CRDTs [1], has been proposed as a middle ground between the two approaches. δ-CRDTs assumes that communication is mostly pairwise, with each replica maintaining a communication buffer for each of its peers where it stores the operations that have not been propagated (and acknowledged) to the remote peer. These buffers are used to compress multiple operations into a single delta, and enforce FIFO communication semantics between each pair of replicas. Whenever a new synchronization path is established between two replicas, the whole state of both replicas has to be synchronized by resorting to a mechanism similar to those employed in State-based CRDTs. Thus, this approach works well, although only in settings with continuous and static synchronization patterns among replicas.

In this paper, we introduce an extension to δ-CRDTs that we name Δ-CRDTs. Δ-CRDTs were specially designed to support dynamic communication patterns among a potentially large number of replicas, and removes the assumption that pairs of replicas are continuously communicating to synchronize their state. Additionally, Δ-CRDTs do not resort to specialized pairwise communication buffers, minimizing the space overhead imposed over each individual replica. Instead, we use the CRDT internal metadata to compute the minimal Delta that needs to be propagated to a remote replica, based on a causal context (usually, a vector clock) that replicas exchange[1]. Due to this, Δ-CRDTs are well suited to be used in decentralized dissemination protocols, such as gossip protocols [4]. To achieve its properties, when

[1]Riak support for big sets uses a similar idea for efficiently identifying removed elements [3].

Algorithm 1: Δ-CRDT replication

```
upon onVersionVector(VV, REPLICA) do
    Δ ⟵ getDelta(vv)
    if Δ.size() > 0
        REPLICA.send(Δ)
    optionally do (push model)
        if vv after self.versionVector
            REPLICA.send(self.versionVector)

upon delta(Δ) do
    self.state.applyDelta(Δ)
    self.versionVector.update(Δ)

periodically do (pull model)
    r ⟵ randomReplica()
    r.send(self.versionVector)

on local operation do (push model)
    r ⟵ randomReplica()
    r.send(self.versionVector)
```

compared to optimized δ-CRDTs, Δ-CRDTs needs to temporarily maintain additional metadata (tombstones). However, this metadata can be garbage collected locally at any time, at the price of being unable to synchronize by sending only a delta when the garbage-collected information is needed for computing the delta. If this happens, the full state needs to be exchanged (as it is always the case when starting a new connection in δ-CRDTs).

We have run a set of preliminary experiments, using a web-based framework, where we compare the performance of our Δ-CRDTs with that of State-based and Operation-based CRDTs, and have observed that our approach enables a better network usage.

2. Δ-CRDTS

Δ-CRDTs are replicated by propagating a delta (Δ) of the current state that is missing in a particular replica. To compute the Δ, a *getDelta* function is called with the causal context of the replica which initiated the communication (i.e., which requires missing updates). This causal context can be sent by a requesting replica (*pull model*) or, when local operations are performed, sent to other replicas (*push model*)[2]. Algorithm 1 shows how a simple replication protocol can be created by leveraging Δ-CRDTs. A replica receives a causal context (version vector) from a replica and computes a Δ that is to be shipped back. A replica can receive a causal context (version vector) from a replica where there is no *is newer than* relationship between the received and its own context (i.e., the relationship is somehow bidirectional). This means that both replicas have executed operations (concurrently) that the other has not yet seen, and thus both a Δ and a causal context have to be shipped (as to ensure the other replica also computes and send a Δ back to that node).

To create Δ-CRDTs, the following methods have to be implemented: a *delta* function must be implemented to be able to compute a Δ from a given point in time (i.e., the causal history, typically in the form of a version vector); a *applyDelta* function must be implemented which applies a given delta to the current state. For some data types, such functions might require to store significant amounts of metadata. Hence these functions should be carefully crafted to avoid such pitfalls.

In the particular case of container like data types, such as Sets and Maps, CRDTs typically associate a unique timestamp to each data-item. To avoid concurrent add-remove anomalies, typically these data-types use a remove-set of unique timestamps, which are called tombstones. In our Δ-CRDTs we use as unique timestamp pairs of *replicaID* and *operationNumber.* This ensures that each existing data-items and tombstones can be related to any given version vector (as to be before or after that point).

Notice that causality is maintained by the same principle associated with shipping the whole state when using State-based CRDTs. *getDelta* always returns the complete Δ and thus all missing operations on the other replica are sent in a single message. A Δ is always added to the local state in a single execution step (i.e., no other methods should be able to access the internal data-structures), and thus causality is implicitly maintained. Note however, that when two replicas are synchronized, or when a replica receives a causal context that is in its future, the generated Δ will be empty.

To be able to compute the delta from a given causal context, Δ-CRDTs need to maintain metadata about deleted elements (note that δ-CRDTs also need to maintain such information in pairwise communication buffers). In order to keep the amount of wasted space small we remove old metadata periodically (i.e., we provide a mechanism to garbage collect old tombstones). A *garbageCollection* function is added which removes old metadata associated with all operations that happened before a given point in time (also denoted by a version vector).

When garbage collection occurs, the previously described *applyDelta* function has to be able to still infer if some portion of the current local state is outdated (i.e., removed data-items whose's tombstones have been garbage collected). The *getDelta* function is adapted to handle the (typically rare) case where the local replica's garbage collection point is further ahead in time than the sender's causal context. In this case, a Δ-CRDT falls back to a State-based CRDT merge procedure, where the whole state, including the causal context of the last garbage collection step, have to be shipped and integrated by the remote replica.

The main drawback of using Δ-CRDTs is expected to be an increase in latency for replicas to receive operations. Typically State-based CRDTs and Operations-based CRDTs use a *push model* to propagate local changes to a replica. Though these data-types are able to immediately send the changes, Δ-CRDTs need an additional communication step between replicas. Typically, a version vector is first sent, and then a delta is sent back which can be locally applied. A version vector can also be piggy-backed along with the delta, as to ensure the initiating replica also ships any locally applied changes that the remote replica has not yet received. When using Δ-CRDTs with stable communication patterns, the additional communication step is paid only when establishing the connection.

When used in a scenario with dynamic communication patterns and compared to δ-CRDTs, Δ-CRDTs have the following advantages: (1) Δ-CRDTs do not require each

[2]The distinction between *pull* and *push* can be found in [4]

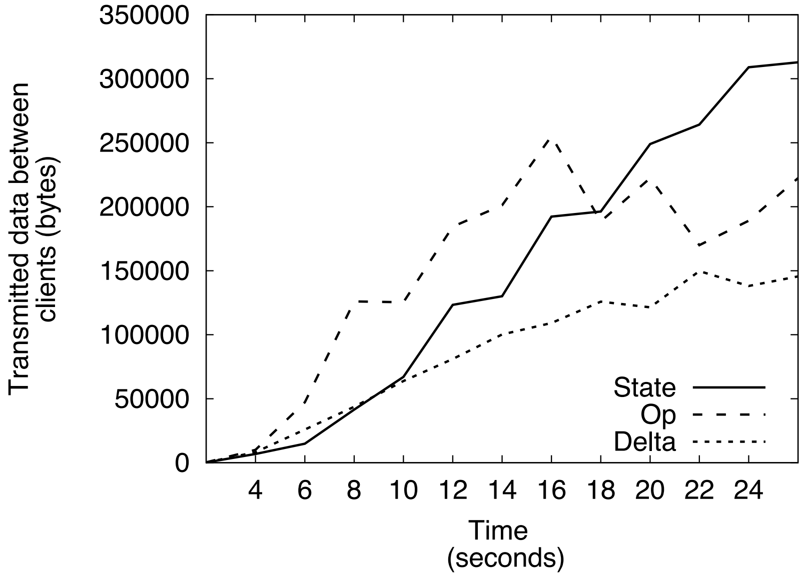

Figure 1: Communication cost, results for 8 replicas sharing an Observe-Remove Set. Each replica issues an operation every 500ms. The workload is composed of 70% inserts and 30% removes.

replica to maintain a buffer for each of its connections; (2) by using the information exchanged in the vector clocks, a replica will only send the minimal Delta needed by the remote replica, instead of sending all the information stored in the Delta (that might have arrived to the remote replica through a different communication path).

3. PRELIMINARY RESULTS

To have an initial feel for the feasibility of Δ-CRDTs in comparison to State-based CRDTs or Operation-based CRDTs in a real setting we compare the usage of each type of CRDT in a peer-to-peer setting. We implemented Δ-CRDTs, State-based CRDTs, and Operation-based CRDTs, namely an Observe-Remove Set, extending an existing browser-based peer-to-peer framework which has support for State-based CRDT and Operation-based CRDT replication.

We run multiple nodes (each node owns a replica of a single replicated set) in a peer-to-peer setting. The interactions between active peers is dynamic, i.e., replicas communicate with a random sub-set of all existing replicas at each synchronization step.

3.1 Implementation

Communication between replicas happens every T seconds. In a synchronization step, a random subset of the currently connected neighbours are selected by a peer. At this point, using Δ-CRDTs, the causal context of the initiating replica is sent to those peers (hence, we use a pull communication model). In contrast, when using State-based CRDTs the whole state is shipped to the randomly selected peers.

When using Operation-based CRDTs the version vector is also shipped. This happens because in our experimental setting there is no continuous flow of messages between pairs of nodes (i.e., the communication patterns change at each synchronization step). The alternative would require each replica in the system to maintain information about all operations which have previously been sent and acknowledged. The remote replica will use this vector clock to send back missing operations (and its own version vector).

Note that while we use a *push model* when propagating State-based CRDTs, a *pull model* is employed for Operation-based CRDTs and Δ-CRDTs.

3.2 Experimental Setup

We run 8 clients in a browser-based peer-to-peer framework where each client continuously issues operations over a replicated CRDT Observe-Remove Set. Each client was ran in its own Google Chrome instance, on a local machine (MacBook Pro Retina, 16GB RAM). All reported results are the mean result of three independent runs.

We compare the sizes of messages sent between clients when using Δ-CRDTs, State-based CRDTs, and Operation-based CRDTs. The Set is updated, by each peer, twice per second. Each peer, per update, has a 30 % change to remove an existing data-item and 70 % change to add a new data-item (a string with 14 characters, with 2 bytes per char resulting in 28 bytes per data-item). Each peer contacts 2 randomly selected peers, every 5 seconds, as to begin state reconciliation between them (as discussed previously).

3.3 Results

Figure 1 reports the obtained results showing the size, in bytes, of all messages exchanged between replicas, with a sampling interval of one second (object related messages only, including state, operations, Δs, and version vectors when applicable). As expected, State-based CRDTs have an always growing load on the network. As more operations are executed more state has to be exchanged between replicas. The currently implemented Operation-based CRDTs are not optimized for the employed communication model and thus incur an initial load penalty. As only operations are sent over the network (along with version vectors), eventually the network load becomes lower than state propagation.

Δ-CRDTs propagate less data over the network as, when the total amount of applied operations increases, what is shipped between clients is always a Δ where this Δ is much smaller that the whole state of the object.

4. CONCLUSION AND FUTURE WORK

The results that we have reported show that, in a scenario with highly dynamic communication patterns, Δ-CRDTs clearly outperform, from the standpoint of network usage, the competing alternative (we are currently updating our browser-based peer-to-peer framework to make use of Δ-CRDTs due to this). We intend to further implement and evaluate Δ-CRDTs in a distributed setting (scenarios with geo-replication for instance). We will also continue working on improving the current implementation of Δ-CRDTs, as the current implementations have unbounded growth on the version vector size (i.e., it is bounded by the amount of replicas in the system which can grow significantly when pushing replicas to the client side).

Acknowledgments

This research is supported in part by EU FP7 SyncFree project (609551) and NOVA LINCS (UID/CEC/04516/2013).

5. REFERENCES

[1] P. S. Almeida, A. Shoker, and C. Baquero. Efficient state-based crdts by delta-mutation. In A. Bouajjani and H. Fauconnier, editors, *Networked Systems - Third International Conference, NETYS 2015, Agadir, Morocco, May 13-15, 2015, Revised Selected Papers*, volume 9466 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 2015.

[2] E. Brewer. Towards robust distributed systems (abstract). In *ACM PODC*, page 7, 2000.

[3] R. Brown. Riak support for big sets (private communication), 2015.

[4] J. Leitão. *Topology Management for Unstructured Overlay Networks*. PhD thesis, Technical University of Lisbon, Sept. 2012.

[5] M. Shapiro, N. M. Preguiça, C. Baquero, and M. Zawirski. Conflict-free replicated data types. In X. Défago, F. Petit, and V. Villain, editors, *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*, volume 6976 of *Lecture Notes in Computer Science*, pages 386–400. Springer, 2011.