J. Parallel Distrib. Comput. 111 (2018) 162–173

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Delta state replicated data types☆

Paulo Sérgio Almeida [1], Ali Shoker *,[1], Carlos Baquero [1]

Departamento de Informática, Universidade do Minho, Campus de Gualtar, 4710-057 Braga, Portugal

HIGHLIGHTS

- Definition of delta-state CRDTs and their relation to state CRDTs.
- Proofs of conditions to attain equivalence to state based CRDTs.
- Anti-entropy algorithm for basic and causally consistent convergence.
- Portfolio of delta state CRDTs including optimized sets, and recursive map.

ARTICLE INFO

Article history:
Received 19 April 2016
Received in revised form 13 June 2017
Accepted 13 August 2017
Available online 18 August 2017

Keywords:
Distributed systems
Eventual consistency
State-based
Delta
Conflict-free replicated data types
CRDT

ABSTRACT

Conflict-free Replicated Data Types (CRDTs) are distributed data types that make eventual consistency of a distributed object possible and non ad-hoc. Specifically, state-based CRDTs ensure convergence through disseminating the entire state, that may be large, and merging it to other replicas. We introduce *Delta State Conflict-Free Replicated Data Types* (δ-CRDT) that can achieve the best of both operation-based and state-based CRDTs: small messages with an incremental nature, as in operation-based CRDTs, disseminated over unreliable communication channels, as in traditional state-based CRDTs. This is achieved by defining δ-*mutators* to return a *delta-state*, typically with a much smaller size than the full state, that to be joined with both local and remote states. We introduce the δ-CRDT framework, and we explain it through establishing a correspondence to current state-based CRDTs. In addition, we present an anti-entropy algorithm for eventual convergence, and another one that ensures causal consistency. Finally, we introduce several δ-CRDT specifications of both well-known replicated datatypes and novel datatypes, including a generic map composition.

1. Introduction

Eventual consistency (EC) is a relaxed consistency model that is often adopted by large-scale distributed systems [15,18,34] where availability must be maintained, despite outages and partitioning, whereas delayed consistency is acceptable. A typical approach in EC systems is to allow replicas of a distributed object to temporarily diverge, provided that they can eventually be reconciled into a common state. To avoid application-specific reconciliation methods, costly and error-prone, *Conflict-Free Replicated Data Types* (CRDTs) [32,33] were introduced, allowing the design of self-contained distributed data types that are always available and eventually converge when all operations are reflected at all replicas. Though CRDTs are deployed in practice and support millions of users worldwide [9,21,30], more work is still required to improve their design and performance.

CRDTs support two complementary designs: *operation-based* (or op-based) and *state-based*. In op-based designs [26,33], the execution of an operation is done in two phases: *prepare* and *effect*. The former is performed only on the local replica and looks at the operation and current state to produce a message that aims to represent the operation, which is then shipped to all replicas. Once received, the representation of the operation is applied remotely using *effect*. On the other hand, in a state-based design [5,33] an operation is only executed on the local replica state. A replica periodically propagates its local changes to other replicas through shipping its entire state. A received state is incorporated with the local state via a *merge* function that deterministically reconciles both states. To maintain convergence, *merge* is defined as a *join*: a least upper bound over a join-semilattice [5,33].

Op-based CRDTs have some advantages as they can allow for simpler implementations, concise replica state, and smaller messages; however, they are subject to some limitations: First, they

☆ The work presented was partially supported by EU FP7 SyncFree project (609551), EU H2020 LightKone project (732505), and SMILES line in project TEC4Growth (NORTE-01-0145-FEDER-000020).

* Corresponding author.
E-mail addresses: psa@di.uminho.pt (P.S. Almeida), shokerali@di.uminho.pt (A. Shoker), cbm@di.uminho.pt (C. Baquero).
[1] HASLab/INESC TEC and Universidade do Minho, Portugal.

http://dx.doi.org/10.1016/j.jpdc.2017.08.003
0743-7315/

assume a message dissemination layer that guarantees reliable exactly-once causal broadcast; these guarantees are hard to maintain since large logs must be retained to prevent duplication even if TCP is used [20]. Second, membership management is a hard task in op-based systems especially once the number of nodes gets larger or due to churn problems, since all nodes must be coordinated by the middleware. Third, the op-based approach requires operations to be executed individually (even when batched) on all nodes.

The alternative is to use state-based systems, which are free from these limitations. However, a major drawback in current state-based CRDTs is the communication overhead of shipping the entire state, which can get very large in size. For instance, the state size of a *counter* CRDT (a vector of integer counters, one per replica) increases with the number of replicas; whereas in a *grow-only Set*, the state size depends on the set size, that grows as more operations are invoked. This communication overhead limits the use of state-based CRDTs to data-types with small state size (e.g., counters are reasonable while large sets are not). Recently, there has been a demand for CRDTs with large state sizes (e.g., in RIAK DT Maps [10] that can compose multiple CRDTs and that we formalize in Section 7.4.9).

In this paper, we rethink the way state-based CRDTs should be designed, having in mind the problematic shipping of the entire state. Our aim is to ship a *representation of the effect* of recent update operations on the state, rather than the whole state, while preserving the idempotent nature of *join*. This ensures convergence over unreliable communication (on the contrary to op-based CRDTs that demand exactly-once delivery and are prone to message duplication). To achieve this, we develop in detail the concept of *Delta State-based CRDTs* (δ-CRDT) that we initially introduced in [2]. In this new (delta) framework, the state is still a join-semilattice that now results from the join of multiple fine-grained states, i.e., *deltas*, generated by what we call δ-*mutators*. δ-*mutators* are new versions of the datatype mutators that return the effect of these mutators on the state. In this way, deltas can be temporarily retained in a buffer to be shipped individually (or joined in groups) instead of shipping the entire object. The changes to the local state are then incorporated at other replicas by joining the shipped deltas with their own states.

The use of "deltas" (i.e., incremental states) may look intuitive in state dissemination; however, this is not the case for state-based CRDTs. The reason is that once a node receives an entire state, merging it locally is simple since there is no need to care about causality, as both states are self-contained (including meta-data). The challenge in δ-CRDT is that individual deltas are now "state fragments" and usually must be causally merged to maintain the desired semantics. This raises the following questions: is merging deltas semantically equivalent to merging entire states in CRDTs? If not, what are the sufficient conditions to make this true in general? And under what constraints causal consistency is maintained? This paper answers these questions and presents corresponding proofs and examples.

We address the challenge of designing a new δ-CRDT that conserves the correctness properties and semantics of an existing CRDT by establishing a relation between the novel δ-mutators with the original CRDT mutators. We prove that eventual consistency is guaranteed in δ-CRDT as long as all deltas produced by δ-mutators are delivered and joined at other replicas, and we present a corresponding simple anti-entropy algorithm. We then show how to ensure causal consistency using deltas through introducing the concept of *delta-interval* and the *causal delta-merging condition*. Based on these, we then present an anti-entropy algorithm for δ-CRDT, where sending and then joining delta-intervals into another replica state produces the same effect as if the entire state had been shipped and joined.

We illustrate our approach through a simple *counter* CRDT and a corresponding δ-CRDT specification. Later, we present a portfolio of several δ-CRDTs that adapt known CRDT designs and also introduce a generic kernel for the definition of CRDTs that keep a causal history of known events and a CRDT map that can compose them. All these δ-CRDT datatypes, and a few more, are available online in a reference C++ library [3]. Our experience shows that a δ-CRDT version can be devised for all CRDTs, but this requires some design effort that varies with the complexity of different CRDTs. This refactoring effort can be avoided for new datatypes by writing all mutations as delta-mutations, and only deriving the standard mutators if needed; these can be trivially obtained from the delta-mutators.

This paper is an extended version of [2], adding the following material: Proofs of conditions to attain equivalence to state based CRDTs; Anti-entropy algorithm for basic convergence; Portfolio of delta state CRDTs including simple compositions and anonymous replicated types (grow only sets, two phase sets, lexicographic pairs (Soundcloud [9]) last-writer-wins sets), named types (positive–negative counters, (Cassandra [16]) lexicographic counters); Kernel for causal CRDTs, with a universal join function; Optimized causal CRDTs (remove-wins sets, (Riak) flags [6]); Recursive map data type for causal CRDTs.

2. System model

Consider a distributed system with nodes containing local memory, with no shared memory between them. Any node can send messages to any other node. The network is asynchronous; there is no global clock, no bound on the time a message takes to arrive, and no bounds on relative processing speeds. The network is unreliable: messages can be lost, duplicated or reordered (but are not corrupted). Some messages will, however, eventually get through: if a node sends infinitely many messages to another node, infinitely many of these will be delivered. In particular, this means that there can be arbitrarily long partitions, but these will eventually heal. Nodes have access to durable storage; they can crash but will eventually recover with the content of the durable storage just before the crash occurred. Durable state is written atomically at each state transition. Each node has access to its globally unique identifier in a set $\mathbb{I}$.

2.1. Notation

We use mostly standard notation for sets and maps, including set comprehension of the forms $\{f(x)|x \in S\}$ or $\{x \in S|Pred(x)\}$. A map is a set of (k, v) pairs, where each k is associated with a single v. Given a map m, $m(k)$ returns the value associated with key k, while $m\{k \mapsto v\}$ denotes m updated by mapping k to v. The domain and range of a map m is denoted by $\mathsf{dom}\ m$ and $\mathsf{ran}\ m$, respectively, i.e., $\mathsf{dom}\ m = \{k|(k, v) \in m\}$ and $\mathsf{ran}\ m = \{v|(k, v) \in m\}$. We use $\mathsf{fst}p$ and $\mathsf{snd}p$ to denote the first and second component of a pair p, respectively. We use $\mathbb{B}$, $\mathbb{N}$, and $\mathbb{Z}$, for the booleans, natural numbers, and integers, respectively; also $\mathbb{I}$ for some unspecified set of node identifiers. Most sets we use are partially ordered and have a least element $\perp$ (the bottom element). We use $A \hookrightarrow B$ for a partial function from A to B; given such a map m, then $\mathsf{dom}\ m \subseteq A$ and $\mathsf{ran}\ m \subseteq B$, and for convenience we use $m(k)$ when $k \notin \mathsf{dom}\ m$ and B has a bottom, to denote $\perp_B$; e.g., for some $m : \mathbb{I} \hookrightarrow \mathbb{N}$, then $m(k)$ denotes 0 for any unmapped key k.

3. A background of state-based CRDTs

Conflict-Free Replicated Data Types [32,33] (CRDTs) are distributed datatypes that allow different replicas of a distributed CRDT instance to diverge and ensures that, eventually, all replicas converge to the same state. State-based CRDTs achieve this through propagating updates of the local state by disseminating the entire state across replicas. The received states are then