Exploring the Journey from Sequential Specifications to Pure Operation-based CRDTs

Luís Filipe Recharte
Faculty of Engineering, University of Porto

Hugo Pacheco
Faculty of Sciences, University of Porto

Carlos Baquero
Faculty of Engineering, University of Porto

ABSTRACT

Conflict-free replicated data types (CRDTs) are abstract data structures devised for replication across multiple nodes. They can be categorized as either state-based, where the state is updated, transmitted, and subsequently merged on other replicas, or operation-based, where the operations are generated locally and dispatched to other replicas. Pure operation-based CRDTs were introduced to remark the simplicity and efficiency of op-based CRDTs. Here, the transmission of information is exclusively done through a reliable causal delivery protocol, and delivered messages are then incorporated into a partially ordered log (PO-log). The pure op-based framework makes the design of CRDTs "almost" generic, but leaves some datatype-dependent reasoning to be manually crafted.

We explore how two recent approaches for the design of CRDTs can be put to use to accomplish "closer to" generic pure op-based CRDTs, where users only reason sequentially, developing a datatype, its operations and semantic properties, and a CRDT consistent with sequential execution is automatically constructed. This is also an exercise on understanding and explaining the distributed design of complex CRDTs from the semantics of sequential datatypes.

KEYWORDS

CRDTs, ECROs, Semidirect Product, Distributed Systems, Availability, Strong Eventual Consistency

1 INTRODUCTION

A CRDT is an abstract data type that enables a system to converge to a common state without synchronization as long as the effect of (concurrent) operations on the state is commutative, i.e., independent of their order [**preguica_conflict-free_2018**]. CRDTs prove highly advantageous in distributed systems where strong consistency models become impractical due to the demanding coordination and communication requirements. By leveraging CRDTs, it becomes possible to safeguard convergence while enhancing availability and scalability, thereby striking a favorable balance between these critical factors.

Motivation. It is trivial to turn a sequential data type into a distributed CRDT if all its (concurrent) operations are commutative: the distributed effect of each operation can be simply defined as applying the sequential semantics of its operations. In practice, this is the case for simple CRDTs such as counters with increment/decrement operations, but in general requires severely restricting data types and their operations, e.g., operations on registers (allowing writes to the counter's value) or sets are not commutative.

The craft of CRDTs can be precisely framed as coming up with a distributed object with commutative effects that mimics a sequential data type with non-commutative operations. As this is often a manual and ingenious task, the community has been proposing various ways to design CRDTs. However, reasoning about CRDTs is classically performed w.r.t. a concurrency semantics, meaning that the behavior of distributed effects is often understood from a global perspective that considers the history of updates over all replicas. In other words, the distributed behavior of CRDTs can not be easily (or even at all) explained from the sequential perspective of a single replica, blurring the connection to the original sequential data type.

The curious case of the RGA. One of the most classical CRDTs is the Replicated Growable Array (RGA), modelling lists. Still, connecting the RGA design to sequential lists is far from trivial. List operations typically identify elements by their indexes. However, `insert` and `delete` operations on lists are not directly commutative, as the index of a list element may change if other elements are inserted or deleted concurrently, leading to inconsistencies.

To increase the independence among list operations, RGA assigns a unique identifier to each element of the list, which is generated in a monotonically increasing way by each local replica. When performing an `insert` operation, the new element is placed in the array according to the uniquely referenced position, taking into account concurrent `inserts` to the same position. Even after a `delete` operation, concurrent inserts can still be referencing the deleted element. Thus, the targeted element is only marked with a "tombstone" to be effectively ignored by future queries. The element is then permanently removed once its `delete` operation stabilizes.

Contributions. In this paper, we will explore recent techniques for building CRDTs from sequential data type specifications, namely Explicitly Consistent Replicated Objects (ECROs) [**de_porre_ecros_2021**] and the Semidirect Product [**weidner_composing_2020**]. We will then present and implement our own general CRDT construction that combines and generalizes upon both techniques. Along the way, we will also shed light into the design of the RGA, and illustrate how its complex behavior can be derived from a sequential list data type using our construction.

2 A PURE OPERATION-BASED FRAMEWORK

Pure operation-based CRDTs [**baquero_pure_2017**] require a communication middleware called a Tagged Causal Stable Broadcast (TCSB), which we have built following [**younes_dynamic_2022**]. A client process connectso the CRDT and the middleware. The middleware provides causality and supplementary information as message meta-data, and also indicates when a message stabilizes, to enable PO-log compaction. We have also built a randomized testing framework to evaluate the performance, convergence, and consistency of developed CRDTs.

3 ECROS

ECROs [**de_porre_ecros_2021**] are a fully generic approach to turn any sequential data type into a distributed CRDT. For that purpose, users can provide a preferential arbitration order among operations. The CRDT construction will then dynamically decide on a global ordering for sequences of operations, and rearrange operations as needed. An ECRO guarantees convergence (all replicas execute the same sequence of operations) and supports stabilization (removing older operations), which is essential for efficiency.

The ECRO approach is independent from the semantics of the data type and its operations. It will ensure convergence by deterministically ordering operations in each replica and rolling back updates when inconsistencies are found. The semantic information about operations – in particular commutativity – is only used as an optimization, to help minimizing the number of rollbacks and the stored metadata for commutative operations.

4 SEMIDIRECT SEQUENCES

A very different approach to constructing CRDTs inspired by Operational Transformation (OT) is put forward by the Semidirect Product [**weidner_composing_2020**]. Unlike ECROs, it leverages the data type semantics to repair operations in a way that ensures convergence without rollback. Naturally, it does so by imposing a more rigid structure on the underlying operations. In particular, the Semidirect Product considers a disjoint partitioning of operations into two classes A and B, and embodies a binary arbitration criterion ($\mathtt{A} < \mathtt{B}$) that prioritizes B operations over A operations, by using a "repair right" transformation function ($\triangleright : \mathtt{B} \to \mathtt{A} \to \mathtt{A}$). For two updates $a \in A$ and $b \in B$, with $\mathtt{a} < \mathtt{b}$, "repairing right" must satisfy the property:

$$a \ ; \ b = b \ ; \ b \triangleright a$$

The intuition is that all replicas will repair operations to appear as if *almost* all A operations were rearranged to be applied before B operations. Unfortunately, the Semidirect Product does not directly scale beyond two classes. We remark that this is precisely because of the "almost" part: it can not repair/rearrange causally dependent operations that go against the arbitration order (when $b \prec a$).

Thus, we propose a more general Semidirect Sequence construction. On one side, we consider that causally-dependent updates always respect the arbitration order. Fortunately, it turns out that RGA `insert` operations precisely satisfy this restriction. On the other side, this restriction allows our much more relaxed Semidirect Sequence construction to repair operations across the continuum offered by a total arbitration order. In particular, for two concurrent updates ($u_1 \mid\mid u_2$), with $u_1 \leq u_2$, we only require that:

$$u_1 \ ; \ u_2 = u_2 \ ; \ u_2 \triangleright u_1$$
$$u_2 \triangleright u_1 \leq u_2$$

This means that operations only repair lower operations regarding the defined arbitration order. Consequently, the operation resulting from this repair process will always be less than or equal to the operation with the higher standing in the arbitration order. In turn, this allows maintaining the log of operations always ordered, guaranteeing that the state of a replica is always equivalent to applying a sequence of causally consistent sequential operations.

5 SEMIDIRECT SEQUENCES AND ECROS

We have captured the behavior of RGA `inserts`. Yet, our Semidirect Sequence can not simultaneously model `deletes`. Even if we postpone `deletes` (`insert` < `delete`), there is no repair such that:

$$\mathtt{insert}\ t\ t' \ ; \ \mathtt{delete}\ t = \mathtt{delete}\ t \ ; \ \mathtt{delete}\ t \triangleright \mathtt{insert}\ t\ t'$$

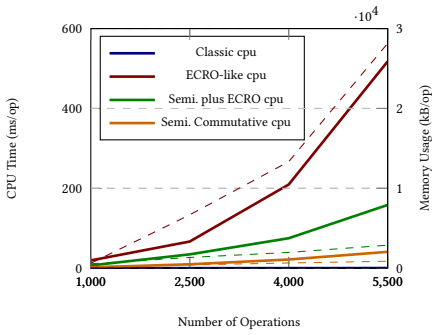
Our path to capture `deletes` is to show that we can extend a Semidirect Sequence of repairable operations with general ECRO operations (Since `deletes` are commutative, a much simpler structure would suffice for the RGA, but we emphasise generality). By placing `inserts` in the Semidirect logic and `deletes` in the ECRO logic, the intuition is that the combined CRDT will keep a state of applied `inserts`, repaired to satisfy the arbitration order, and `deletes` will be applied at the end of the state, with eventual rollbacks.

The classical RGA assumes that elements are always inserted after existing positions, which avoids updates against the arbitration order ($\mathtt{delete}\ t \prec \mathtt{insert}\ t\ t'$), but forces each replica to check its state before issuing `insert` operations. In general, and to support insertions after nonexistent positions, we introduce a new "repair left" transformation function ($\triangleleft : \mathtt{A} \to \mathtt{B} \to \mathtt{A}$) that, given $a \in A$ and $b \in B$, with $b \prec a$, can be defined such that:

$$b \ ; \ a = a \triangleleft b \ ; \ b$$

6 APPLICATIONS: RGA AND BEYOND

We conduct a comparative performance analysis of our implementation across four RGA constructions in a system with 5 replicas. The Semidirect Sequences plus ECRO approach outperforms the ECRO approach alone. The combined approach benefits from maintaining a smaller log of operations, with less computation and fewer rollbacks. Meanwhile, since `deletes` are commutative and computation inexpensive, we consider a simpler combined approach that does not keep a log and has further improved performance, by delaying `deletes` to query time. Naturally, the classical RGA implementation outperforms the generic constructions.



We remark that the causal restrictions of our Semidirect Sequences are sufficient but not necessary for convergence, which makes us believe that further relaxations – to model more complex CRDTs – would be possible. Another path to explore is the automated synthesis of highly optimized CRDTs from our constructions.