Efficient (re)naming in Conflict-free Replicated Data Types (CRDTs)

Matthieu Nicolas
matthieu.nicolas@inria.fr

Inria, F-54600, Université de Lorraine, LORIA, F-54506, CNRS, F-54506

1 Abstract (maximum of 100 words).

TODO

2 Clear statement of the identified research problem(s) and the context in which the problem(s) will be addressed.

In order to serve an ever-growing number of users and provide an increasing volume of data, large scale systems such as data stores or collaborative editing tools have to adopt a distributed architecture. However, as stated by the CAP theorem, such systems cannot ensure both strong consistency and high availability in case of network partitions. As a result, literature and companies increasingly adopt the optimistic replication model paired with the eventual consistency model to replicate data among nodes. This consistency model allows replicas to temporarily diverge to be able to ensure high availability, even in case of network partition. Each node owns a copy of the data and can edit it, before propagating updates to others. A conflict resolution mechanism is however required to handle updates generated concurrently by different replicas.

An approach, which gains in popularity since a few years, proposes to define Conflict-free Replicated Data Types (CRDTs). These data structures behave as traditional ones, like the *Set* or *Sequence* data structures, but are designed for a distributed usage. Their specification ensures that concurrent updates are resolved deterministically, without requiring any kind of agreement, and that replicas eventually converge immediately after observing some set of updates, thus achieving *Strong Eventual Consistency.*

To achieve convergence, CRDTs proposed in the literature mostly rely on unique identifiers to reference updated elements. To generate such element identifiers, nodes often use their own identifier as well as a logical clock. Thus, regarding to how node identifiers are generated, the size of element identifiers usually increases with the number of nodes. Furthermore, element identifiers have to comply to additional constraints according to the CRDT, for example forming a dense set in case of a *Sequence* data structure. In this case, element identifiers' size also increases according to the updates performed on the data structure. Therefore, the size of element identifiers is usually not bounded.

Since the size of identifiers attached to each element is not bounded, the overhead of the replicated data structure increases over time. Since nodes have to store and broadcast the identifiers, the application's performances and efficiency decrease over time. This impedes the adoption of concerned CRDTs.

We present an approach to address the issue of the growth of identifiers for a family of CRDTs suffering particularly from this issue : *Sequence* CRDTs.

3 Summary (with appropriate references) of the state-of-the-art related to the identified problem(s) along with a clear restatement of the “gap” relative to the research problem(s).

- Several CRDTs allow to replicate sequences : Treedoc, Logoot, LogootSplit
- To ensure convergence, attach an identifier to each inserted element
- Identifiers allow to identify an element and order them
- Thus identifiers have to comply to several constraints : globally unique, totally ordered and belong to dense set
- Because of these constraints, grow quicker

The *Sequence* data structure represents a numbered suite of values. Two operations are usually defined to update the sequence : *insert(index, element)* adds the *element* at the given *index* whereas *delete(index)* removes the element at the given *index*.

TODO: Introduce *Sequence* CRDTs: Treedoc, Logoot, LogootSplit, ...
Over the years, several CRDTs were proposed to represent replicable sequences. To ensure convergence, these CRDTs generate identifiers they attach to elements. These identifiers have several purposes : they uniquely identify each element, easing their later deletion. The identifiers must then be globally unique. The identifiers are also used to represent the positions of the elements. By comparing their respective identifiers, we are then able to order each element relatively to others. Identifiers must then be totally ordered. Finally, we should always be able to insert a new element between two others. It means that we should always be able to generate a new identifier between two others. Therefore the identifiers should belong to a dense set. This last property is usually address by designing identifiers of variable length.

To address the issue of evergrowing identifiers, several approaches were presented since then. In LSEQ, authors propose several strategies to generate identifiers and design a mechanism to switch between these strategies to limit the speed of the identifiers growth. In Core-Nebula, authors propose a renaming mechanism to reduce the size of currently used identifiers, based on a consensus algorithm. To prevent a system-wide consensus, they divide the system into two tiers : the *Core*, a small set of controlled and stable nodes, and the *Nebula*, an uncontrolled set of nodes. The *Core* ... while a catch-up mechanism is provided for nodes from the *Nebula* to transform the updates they performed concurrently to a renaming.

In our approach, we propose a fully distributed renaming mechanism, allowing any node to perform a renaming without any kind of coordination with others.

4 Statement explaining the approach and results, according to PhD stage: state the intended approach including a summary of work accomplished to date (if any).

We define a new operation : *rename*. This operation generates a new, equivalent state of the current sequence but with arbitrary, shorter identifiers. A map, called *renamingMap*, is also generated to link each identifier from the former sequence to the corresponding identifier in the new one. By propagating this map, other nodes are able to apply the same renaming to their own local copy.

However, being a distributed system, nodes can perform updates and broadcast them while another node is performing concurrently the renaming process.

Upon the reception of these concurrent updates, the node which triggered the renaming process cannot apply them to its copy. Indeed, the identifiers having changed, performing the updates would not insert the elements at the correct positions nor delete the correct elements.

To address this issue, we designed rewriting rules for concurrent operations to a *rename*. By using the original identifier and the *renamingMap*, we are able either to find the new corresponding identifier, if it has been renamed, either to deterministically generate a new identifier preserving the existing order, if it has been concurrently inserted.

However, one update a node can perform and broadcast while another node is performing concurrently the renaming process is actually another *rename*. We thus have to be able to deal with concurrent *rename* operations. To solve this issue, we define a total order between *rename* operations. TODO: Explain that we discard but one *rename* operation. All "losing" operations are undone using additional rewriting rules which also generate new identifiers preserving the order for identifiers inserted after a renaming.

Based on this approach, we propose a new version of the LogootSplit algorithm implementing this renaming mechanism. We also introduce an optimisation to reduce the bandwidth consumption of the *rename* operation by compressing the *renamingMap*.

5 Description of evaluation : describe the evaluation plan including intended metrics (quantitative and/or qualitative).

The designed renaming mechanism has been implemented in MUTE, the collaborative text editor developed by our research team. It is now in the process of being tested.

However, testing does not ensure correctness. We are thus taking steps to provide a formal proof of the algorithm, either using a proof assistant either using a automatic theorem prover.

In parallel, we are designing a benchmarker for collaborative editing tools. The goal is be able to replay a collaborative editing session, from its logs, using different conflicts resolution mechanisms to compare their respective performances. We intend to mesure common metrics like the memory and CPU usage, bandwith consumption but also more specific metrics like the time to propagate and integrate an update and the global time required to converge.

6 Conclusion: that includes a statement of the real or potential impact(s) of solving the identified research problem(s). Finishers especially should include a brief conjecture about future work that builds on the PhD research.

- With renaming mechanism, bound the size of identifiers
- Allow new usages by products for which unbounded size is a no-go like distributed databases
- Future works : propose new strategies to trigger the renaming process to achieve better performances
- Future works : propose new strategies to pick which *rename* operation wins

7 Acknowledgements that properly recognize others' contributions to the work (supervisor(s), other graduate students, funding sources, etc.).