Efficient renaming in Conflict-free Replicated Data Types (CRDTs)

Matthieu Nicolas
Université de Lorraine, CNRS, Inria, LORIA
F-54500, France
matthieu.nicolas@inria.fr

Abstract

Sequence Conflict-free Replicated Data Types (CRDTs) allow to replicate and edit, without any kind of coordination, sequences in distributed systems. To ensure convergence, existing works from the literature add metadata to each element but they do not bound its footprint, which impedes their adoption. Several approaches were proposed to address this issue but they do not fit a fully distributed setting. In this paper, we present our ongoing work on the design and validation of a fully distributed renaming mechanism, setting a bound to the metadata's footprint. Addressing this issue opens new perspectives of adoption of these CRDTs in distributed applications.

ACM Reference format:
Matthieu Nicolas. 2018. Efficient renaming in CRDTs. In *Proceedings of Middleware'18, Rennes, France, December 2018,* 2 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 Research Statement

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 adopt a distributed and replicated architecture. However, as stated by the CAP theorem [2], such systems cannot ensure both strong consistency and high availability in presence of network partitions. As a result, literature and companies increasingly adopt an optimistic replication model [7] paired with the eventual consistency model to replicate data among nodes. This consistency model allows replicas to temporarily diverge to ensure high availability, even in case of network partition. Each node owns a copy of the data and can modify it before propagating updates to others. A conflict resolution mechanism is however required to handle updates generated concurrently by different nodes.

An approach, which gains in popularity since a few years, proposes to define Conflict-free Replicated Data Types (CRDTs) [9]. These data types behave as traditional ones, such as the *Set* or *Sequence* data type, 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* [9].

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 combined with a logical clock. Thus, regarding how node identifiers are generated, the size of element identifiers usually increases with the number of nodes. Element identifiers also have to comply to additional constraints according to the CRDT, for instance forming a dense set in the case of a *Sequence* data type. Consequently, the size of element identifiers also increases according to the number of 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 these identifiers, the application's performances and efficiency decrease over time. This severely impedes the adoption of concerned CRDTs.

In this thesis, we aim to address this issue of the growth of identifiers for a specific category of CRDTs suffering particularly from this issue: *Sequence* CRDTs.

2 Related Work

The *Sequence* data type represents an ordered collection elements.

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*.

This data type is often used in applications and, over the years, several CRDTs [8] were proposed to represent replicated sequences, which can be divided into two approaches.

The first one [3] proposes to add to each element a timestamp and a reference to its predecessor. Using these data, a total order between elements is defined. However this approach requires to keep tombstones for deleted elements to handle concurrent insertions.

The second approach [1], on which we focus from this point, proposes to attach an identifier to each inserted element. These identifiers are used to achieve commutative updates by uniquely identifying each element and ordering them relatively to each other.

The downside of this approach is the increasing size of identifiers. Since identifiers are used to order elements, they have to form a dense set so that nodes are always able to insert a new element between two others. However, two identifiers of the same size can be contiguous. When inserting a new element between two such identifiers, there is no other choice than to increase the size of the generated identifier to be able to generate one respecting the intended order. This leads to a specification of identifiers which does not bound their size.

Several approaches were proposed in the literature in order to address the issue of ever-growing identifiers. Nédelec et al. [6] propose several strategies to generate identifiers and design a mechanism to switch between these strategies to limit the speed of the identifiers growth. Leția et al. [4] propose a renaming mechanism to reduce the size of currently used identifiers, relying on a consensus algorithm. To prevent a system-wide consensus, authors divide the system into two tiers: the *Core*, a small set of controlled and stable nodes, and the *Nebula*, an uncontrolled set of nodes. Both can perform updates but only the members of the *Core* participate in the consensus leading to a rename. A catch-up mechanism is provided for the *Nebula* to transform the updates performed concurrently.

Middleware'18, Rennes, France
2018. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

Nonetheless, using consensuses is expensive and not suited to all kinds of applications. In a fully distributed application, there is no central authority to provide a set of stable nodes acting as the *Core*. A efficient distributed renaming mechanism is thus needed.

3 Proposed Approach

We propose a fully distributed renaming mechanism, allowing any node to perform a renaming without any kind of coordination with others. The goal of such mechanism is to reassign shorter identifiers to each current element of the data structure to reduce its footprint.

We define a new operation : *rename*. This operation generates a new, equivalent state of the current sequence but with arbitrary identifiers of minimal size. 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. The *renamingMap* is only used for renaming, it can thus be garbage collected once the corresponding *rename* operation has been observed by every node.

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, since the identifiers have been modified, 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 either able to find the new corresponding identifier, if it has been renamed, ~~either~~ or able to deterministically generate a new identifier preserving the existing order, if it has been concurrently inserted. The same set of rules can be used by nodes to compute their locale state upon the reception of *rename* operations.

However, one *rename* operation can be concurrent to another one. To solve this scenario, we give priority to one operation over the others. Nodes which observed and applied one of the "losing" *renaming* operations have to undo its effect before applying the "winning" one. Thus we designed additional rewriting rules which reverse the effect of a previously applied *rename* while also generating new identifiers preserving the order for identifiers generated after the renaming. To determine which *rename* operation to proceed with, we define and use a total order between *rename* operations.

Based on this approach, we propose a new version of the LogootSplit [1] algorithm implementing this renaming mechanism. LogootSplit is the state of art of Identifier-based Sequence CRDTs. It reduces the footprint of the data structure by aggregating elements into blocks. The renaming mechanism, paired to LogootSplit, allows us to reassign shorter identifiers to elements, but also to merge existing blocks into one, reducing even more the metadata used. [1] We also introduce an optimisation to reduce the bandwidth consumption of the *rename* operation by compressing the *renamingMap*.

4 Ongoing Validation

The proposed renaming mechanism has been implemented in MUTE [5], the real time peer-to-peer collaborative text editor developed by our research team. It is now in the process of being tested.

[1] For further details on the proposed approach, please read https://bit.ly/2yHkdmA

We are also taking steps to provide a formal specification of the algorithm and prove its correctness, either using a proof assistant either using an automatic theorem prover.

Although the renaming mechanism is bound to reduce the memory usage of the data structure over time, we need to assess that the additional computations do not degrade the overall performances. We are thus designing a benchmarker for collaborative editing tools. The goal is to replay a collaborative editing session, from its logs, using different conflicts resolution mechanisms to compare their respective performances. We intend to measure common metrics like the memory and CPU usage, bandwidth consumption but also more specific ones like the time to propagate and integrate an update and the global time needed to converge.

5 Conclusion and Future Work

The proposed renaming mechanism allows us to set a bound to the size of the identifiers. By bounding the size of identifiers, we open new perspectives for the adoption of concerned CRDTs in systems like distributed databases.

Nonetheless, we are still working on several aspects of the proposed renaming mechanism. The strategy used to trigger a renaming has yet to be designed and evaluated. It should keep the size of identifiers low while preventing many *rename* operations to be generated concurrently to achieve good performances. We could also rework the strategy used to pick a "winning" operation from a set of concurrent *rename* operations. The current one, defining and using a total order between these operations, can lead to cases in which nodes have to undo many *rename* operations because of only one concurrent operation. This could provoke performance issues. New strategies could be proposed to reduce the likelihood of these scenarios and improve the overall performances of the system.

References

[1] Luc André, Stéphane Martin, Gérald Oster, and Claudia-Lavinia Ignat. 2013. Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2013*. IEEE Computer Society, Austin, TX, USA, 50–59. DOI: https://doi.org/10.4108/icst.collaboratecom.2013.254123

[2] Eric A. Brewer. 2000. Towards Robust Distributed Systems. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing (PODC '00)*. ACM, New York, NY, USA, 7–. DOI: https://doi.org/10.1145/343477.343502

[3] Loïck Briot, Pascal Urso, and Marc Shapiro. 2016. High Responsiveness for Group Editing CRDTs. In *ACM International Conference on Supporting Group Work*. Sanibel Island, FL, United States. DOI: https://doi.org/10.1145/2957276.2957300

[4] Mihai Leția, Nuno Preguiça, and Marc Shapiro. 2009. *CRDTs: Consistency without concurrency control*. Research Report RR-6956. INRIA. 16 pages. https://hal.inria.fr/inria-00397981

[5] Matthieu Nicolas, VIctorien Elvinger, Gérald Oster, Claudia-Lavinia Ignat, and François Charoy. 2017. MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor. In *ECSCW 2017 - 15th European Conference on Computer-Supported Cooperative Work (Proceedings of 15th European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos)*, Vol. 1. EUSSET, Sheffield, United Kingdom, 1–4. DOI: https://doi.org/10.18420/ecscw2017_p5

[6] Brice Nédelec, Pascal Molli, Achour Mostéfaoui, and Emmanuel Desmontils. 2013. LSEQ: an adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 ACM Symposium on Document Engineering (DocEng 2013)*. 37–46. DOI: https://doi.org/10.1145/2494266.2494278

[7] Yasushi Saito and Marc Shapiro. 2005. Optimistic Replication. *ACM Comput. Surv.* 37, 1 (March 2005), 42–81. DOI: https://doi.org/10.1145/1057977.1057980

[8] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA. 50 pages. https://hal.inria.fr/inria-00555588

[9] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011)*. 386–400. DOI: https://doi.org/10.1007/978-3-642-24550-3_29