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

Matthieu Nicolas

Université de Lorraine, CNRS, Inria, LORIA

F-54500, France

matthieu.nicolas@inria.fr

Abstract

TODO: Write abstract – Matthieu

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 [5] 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 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) [7]. 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* [7].

TODO: Rework the following paragraph – Matthieu 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 have also 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 severly 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. *pour la petite blague, il semble qu'une sequence ne peut se modifier que par la tête ou la queue pas par le milieu (sinon c'est un array) ;) – Gerald*

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 [1, 6, 8] were proposed to represent replicated sequences. To ensure convergence, these data structures attach an identifier to each inserted element. These identifiers are used to achieve transaction-less and commutative updates by uniquely identifying each element and ordering them relatively to each others.

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 *expected? – Gerald* 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 evergrowing identifiers. Nédelec et al. [4] 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. [3] 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 to a renaming.

TO REVIEW: Ajouter une phrase sur le fait qu'effectuer un consensus est coûteux, surtout dans un système très dynamique, pour motiver la conception d'un mécanisme de renommage entièrement distribué – Matthieu Nonetheless, using consensuses is expensive and is not suited to all kind of application. In a fully distributed application, there is no central authority able to provide a set of

stable nodes acting as the *Core*. A fully distributed and efficient renaming mechanism is thus needed.

3 Proposed Approach

TO REVIEW: ajouter quelques mots pour donner l'idée générale de l'approche proposée: renommage des identifiants – Gerald 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.

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 either able to find the new corresponding identifier, if it has been renamed, either able to deterministically generate a new identifier preserving the existing order, if it has been concurrently inserted. Upon the reception of a *rename* operation, nodes can apply the same rules to compute their new local state.

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.

NOTE: Mentionner qu'une renamingMap *n'est utilisée que pour renommer et qu'elle peut donc être transférée en BDD à un moment ? (causal stability ou sinon simple timeout sans l'utiliser) – Matthieu*

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

4 Ongoing Validation

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.

TO REVIEW: Add introduction to explain that we need to assess the performance of the mechanism – Matthieu 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 of the application. We are thus 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.

5 Conclusion and Future Work

TODO: Write conclusion – Matthieu

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

[4] 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

[5] 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

[6] Marc Shapiro and Nuno Preguiça. 2007. *Designing a commutative replicated data type.* Research Report RR-6320. INRIA. https://hal.inria.fr/inria-00177693

[7] 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

[8] Stéphane Weiss, Pascal Urso, and Pascal Molli. 2009. Logoot : A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009.* IEEE Computer Society, Montreal, QC, Canada, 404–412. DOI : https://doi.org/10.1109/ICDCS.2009.75

add references – Matthieu