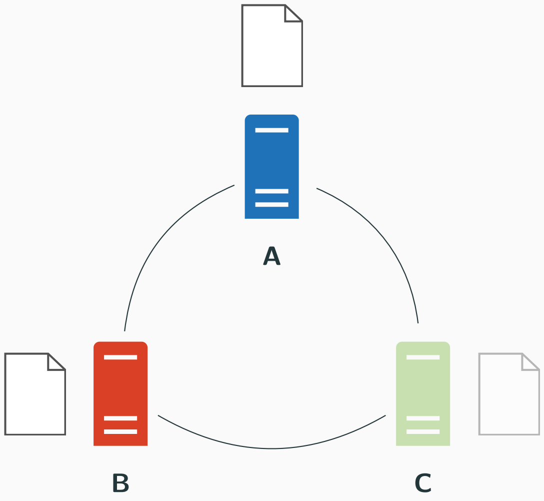
Efficient renaming in Conflict-free Replicated Data Types (CRDTs)

Case Study of a Sequence CRDT : LogootSplit

Matthieu Nicolas (`matthieu.nicolas@inria.fr`)
COAST team
Supervised by Gérald Oster and Olivier Perrin

December 10, 2018

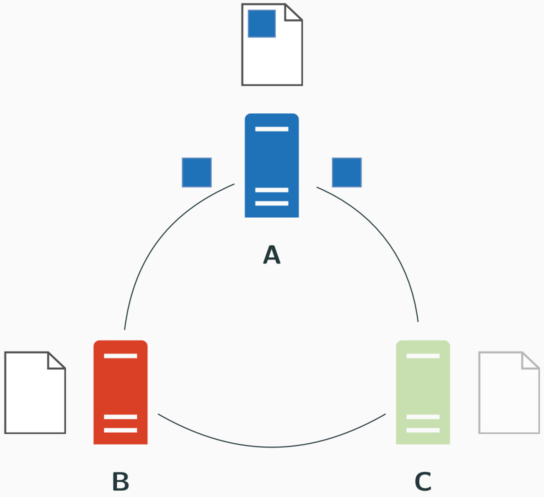
Conflict-free Replicated Data Types (CRDTs)[1]



- Replicated data structure

[1]Marc Shapiro et al. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011 .

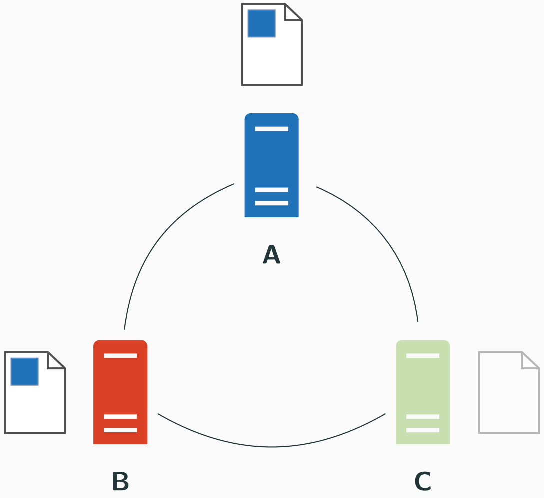
Conflict-free Replicated Data Types (CRDTs)[1]



- Replicated data structure
- Updates performed without coordination

[1]Marc Shapiro et al. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011 .

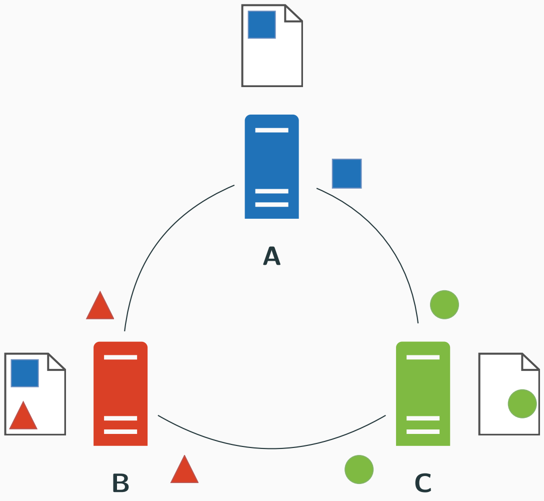
Conflict-free Replicated Data Types (CRDTs)[1]



- Replicated data structure
- Updates performed without coordination

[1]Marc Shapiro et al. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011 .

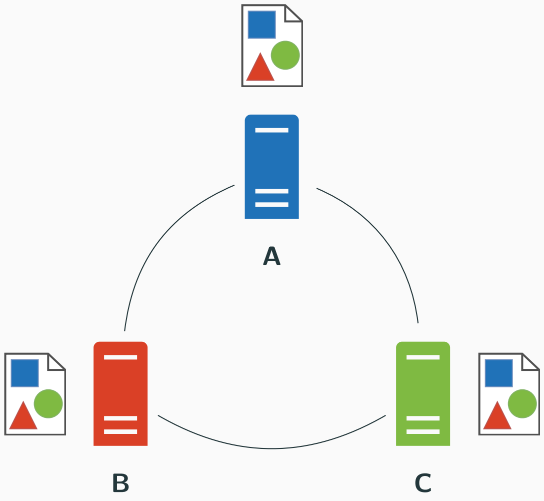
Conflict-free Replicated Data Types (CRDTs)[1]



- Replicated data structure
- Updates performed without coordination

[1]Marc Shapiro et al. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011 .

Conflict-free Replicated Data Types (CRDTs)[1]



- Replicated data structure
- Updates performed without coordination
- Strong Eventual Consistency

[1]Marc Shapiro et al. Conflict-free replicated data types. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 2011 .

Identifier-based CRDTs

Main idea

- Attach an identifier to each element

Allow to design commutative updates

- Identifying uniquely elements
- Ordering updates causally
- ...

Research issue

Limits

- Unbounded size of identifiers
- Overhead of the data structure increasing over time

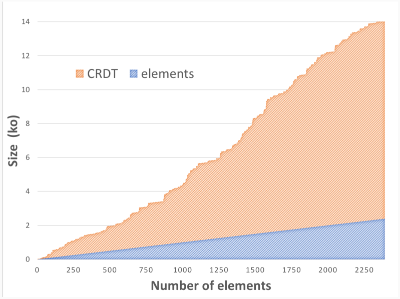


Figure 1: Evolution of the footprint of the data structure

How to reduce the overhead introduced by the data structure ?

How to reduce the overhead introduced by the data structure ?

Reassign shorter identifiers in a fully distributed manner

LogootSplit[2]

- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here as lowercase letters

[2]Luc André et al. Supporting adaptable granularity of changes for massive-scale collaborative editing. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2013*, 2013 .

LogootSplit[2]

- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here as lowercase letters

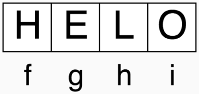


Figure 2: State of a sequence which contains the elements "helo" and their corresponding identifiers

[2]Luc André et al. Supporting adaptable granularity of changes for massive-scale collaborative editing. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2013*, 2013 .

LogootSplit[2]

- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here as lowercase letters

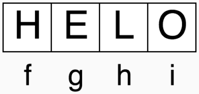


Figure 2: State of a sequence which contains the elements "helo" and their corresponding identifiers

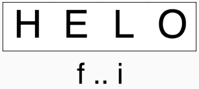


Figure 3: State of a sequence which contains the block "helo"

[2]Luc André et al. Supporting adaptable granularity of changes for massive-scale collaborative editing. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2013*, 2013 .

Example

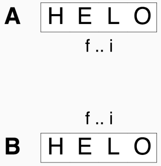


Figure 4: Example of concurrent *insert* operations

Example

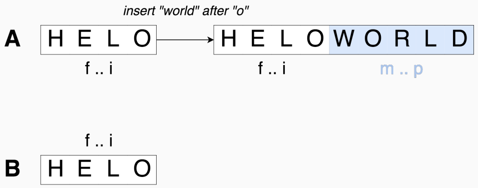


Figure 4: Example of concurrent *insert* operations

Example

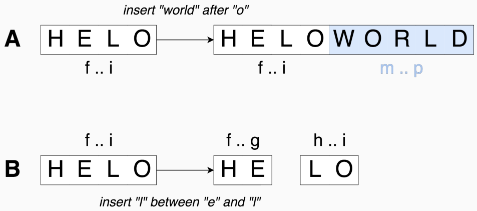


Figure 4: Example of concurrent *insert* operations

Example

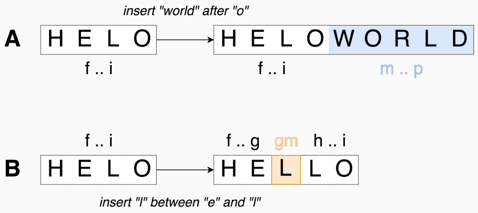


Figure 4: Example of concurrent *insert* operations

Example

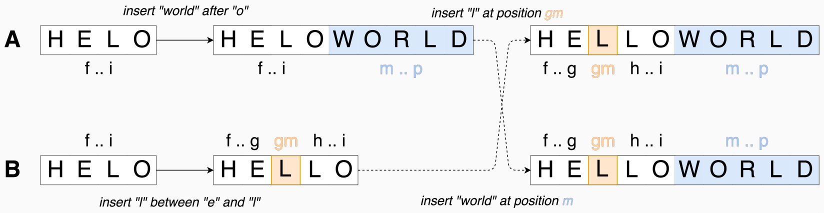


Figure 4: Example of concurrent *insert* operations

Declining performances

- Updates performed may lead to an inefficient internal representation

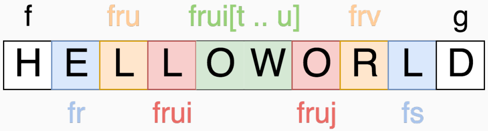


Figure 5: Example of inefficient internal representation

- The more blocks we have:
  - The more metadata we store
  - The longer it takes to browse the sequence to *insert* or *delete* an element

Renaming mechanism

- Introduce a *rename* operation
- Can be perform without coordination

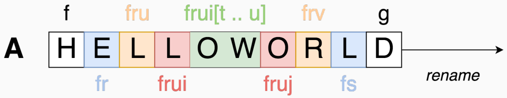


Figure 6: Example of renaming

Renaming mechanism

- Introduce a *rename* operation
- Can be perform without coordination

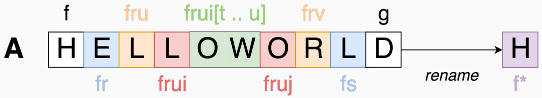


Figure 6: Example of renaming

- Generates a new identifier to the first element, based on its previous identifier

Renaming mechanism

- Introduce a *rename* operation
- Can be perform without coordination

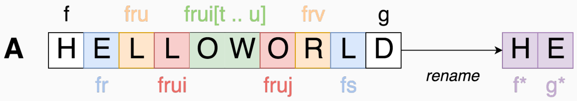


Figure 6: Example of renaming

- Generates a new identifier to the first element, based on its previous identifier
- Then generates contiguous identifiers for all following elements

Renaming mechanism

- Introduce a *rename* operation
- Can be perform without coordination

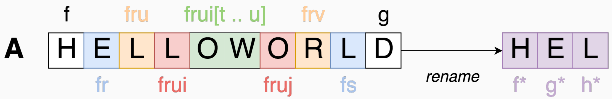


Figure 6: Example of renaming

- Generates a new identifier to the first element, based on its previous identifier
- Then generates contiguous identifiers for all following elements

Renaming mechanism

- Introduce a *rename* operation
- Can be perform without coordination

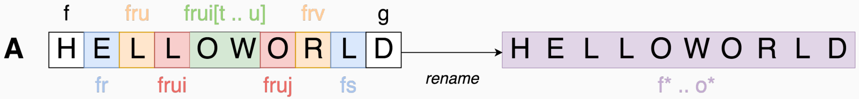


Figure 6: Example of renaming

- Generates a new identifier to the first element, based on its previous identifier
- Then generates contiguous identifiers for all following elements

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation

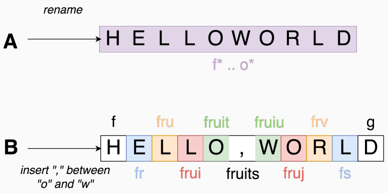


Figure 7: Example of concurrent insert

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation

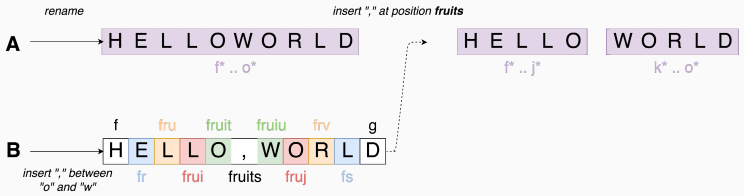


Figure 7: Example of concurrent insert

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation

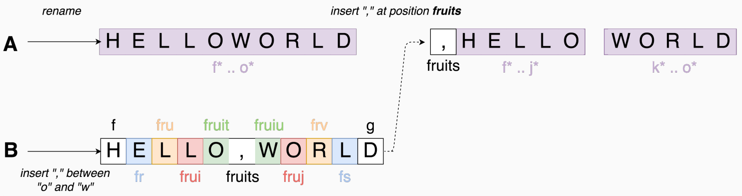


Figure 7: Example of concurrent insert

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation
- May lead to inconsistencies

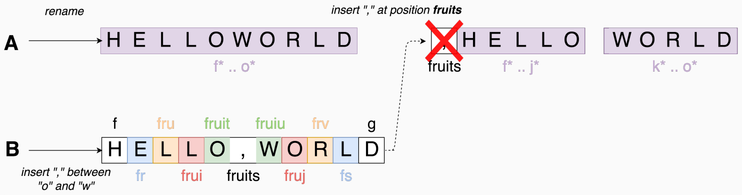


Figure 7: Example of concurrent insert

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation
- May lead to inconsistencies

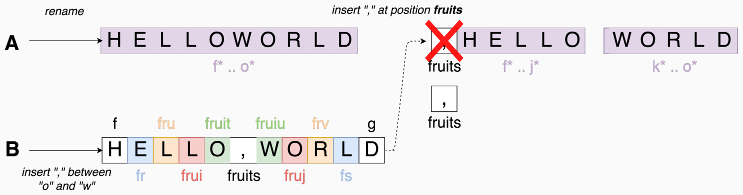


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations

Handling concurrent operations

- Others may perform updates concurrently to a *rename* operation
- May lead to inconsistencies

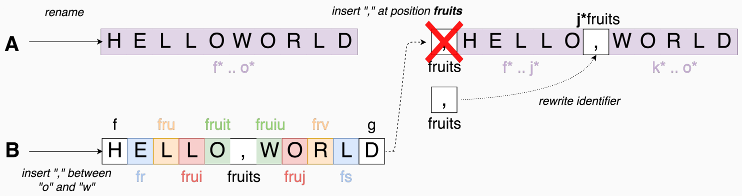


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations
- Define rewriting rules to transform identifiers from one *epoch* to another

What about concurrent rename operations ?

Handling concurrent rename

rename operation not commutative

Handling concurrent rename

rename operation not commutative

- Define a total order between *rename* operations
- Pick a "winner" operation between concurrent *renames*
- Define additional rewriting rules to *undo* the effect of "losing" ones

To wrap up

Done

- Designed a *rename* operation for LogootSplit
- Defined rewriting rules to deal with concurrent updates

[3]Matthieu Nicolas et al. MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor. In Proceedings of European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos, 2017 .

To wrap up

Done

- Designed a *rename* operation for LogootSplit
- Defined rewriting rules to deal with concurrent updates

Work in progress

- Implementing in MUTE[3], our P2P collaborative text editor
- Designing the strategy to trigger the renaming

[3]Matthieu Nicolas et al. MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor. In Proceedings of European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos, 2017 .

To wrap up

Done

- Designed a *rename* operation for LogootSplit
- Defined rewriting rules to deal with concurrent updates

Work in progress

- Implementing in MUTE[3], our P2P collaborative text editor
- Designing the strategy to trigger the renaming

To do

- Prove formally the correctness of the mechanism
- Benchmark its performances (Memory, CPU, Bandwidth,...)

[3]Matthieu Nicolas et al. MUTE: A Peer-to-Peer Web-based Real-time Collaborative Editor. In Proceedings of European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos, 2017 .

Perspectives

Propose a smarter strategy to choose the ”winning” renaming

Perspectives

Propose a smarter strategy to choose the ”winning” renaming

Generalize the approach

- To other Sequence CRDTs
- To other types
    - Counter
    - Set
    - ...

Thanks for your attention, any questions?

LogootSplit identifiers

- To comply with these constraints, LogootSplit proposes identifiers composed of quadruplets of integers of the following form:

$$< \mathit{priority}, \mathit{siteId}, \mathit{seq}, \mathit{offset} >$$

- *priority* allows to determine the position of this identifier compared to others
- *siteId* refers to the node's identifier, assumed to be unique
- *seq* refers to the node's logical clock, which increases monotonically with local operations
- *offset* refers to the element position in its original block

Identifier constraints

- To fulfill their role, identifiers have to comply to several constraints:

Globally unique

- Identifiers should never be generated twice, neither by different users nor by the same one at different times

Totally ordered

- We should always be able to compare and order two elements using their identifiers

Dense set

- We should always be able to add a new element, and thus a new identifier, between two others