

# Ré-identification sans coordination dans les types de données répliquées sans conflits

---

Matthieu Nicolas ([matthieu.nicolas@loria.fr](mailto:matthieu.nicolas@loria.fr))

20 décembre 2022

|                      |                                 |   |
|----------------------|---------------------------------|---|
| <i>Rapporteurs :</i> | Hanifa Boucheneb<br>Davide Frey | Professeure, Polytechnique Montréal<br>Chargé de recherche, HdR, Inria Rennes Bretagne-Atlantique                 |
| <i>Examineurs :</i>  | Hala Skaf-Molli<br>Stephan Merz | Maîtresse de conférences, HdR, Nantes Université, LS2N<br>Directeur de Recherche, Inria Nancy - Grand Est         |
| <i>Encadrants :</i>  | Olivier Perrin<br>Gérald Oster  | Professeur des Universités, Université de Lorraine, LORIA<br>Maître de conférences, Université de Lorraine, LORIA |

State of the art of *Sequence Conflict-free Replicated Data Types (CRDTs)*

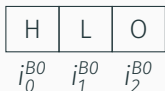
Elements are ordered by their identifier, noted here with the following formalism :  $position_{\substack{node\_id \\ offset}}^{node\_seq}$

---

1. 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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, octobre 2013. DOI : [10.4108/icst.collaboratecom.2013.254123](https://doi.org/10.4108/icst.collaboratecom.2013.254123) .

State of the art of *Sequence CRDTs*

Elements are ordered by their identifier, noted here with the following formalism :  $position_{offset}^{node\_id \ node\_seq}$

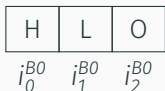


**FIGURE 1** – State of a sequence which contains the elements "HLO" and their corresponding identifiers

- 
1. 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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, octobre 2013. DOI : [10.4108/icst.collaboratecom.2013.254123](https://doi.org/10.4108/icst.collaboratecom.2013.254123) .

State of the art of *Sequence CRDTs*

Elements are ordered by their identifier, noted here with the following formalism :  $position_{offset}^{node\_id \ node\_seq}$



**FIGURE 1** – State of a sequence which contains the elements "HLO" and their corresponding identifiers

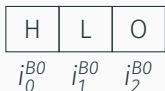


**FIGURE 2** – State of a sequence which contains the block "HLO"

- 
1. 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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, octobre 2013. DOI : [10.4108/icst.collaboratecom.2013.254123](https://doi.org/10.4108/icst.collaboratecom.2013.254123) .

## State of the art of *Sequence CRDTs*

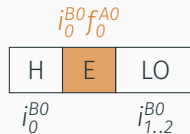
Elements are ordered by their identifier, noted here with the following formalism :  $position_{offset}^{node\_id \ node\_seq}$



**FIGURE 1** – State of a sequence which contains the elements "HLO" and their corresponding identifiers



**FIGURE 2** – State of a sequence which contains the block "HLO"



**FIGURE 3** – State of a sequence which contains the elements "HELO" and their corresponding identifiers

- 
1. 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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, octobre 2013. DOI : [10.4108/icst.collaboratecom.2013.254123](https://doi.org/10.4108/icst.collaboratecom.2013.254123) .

# 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

# LogootSplit identifiers

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

$$position_{offset}^{node\_id \ node\_seq}$$

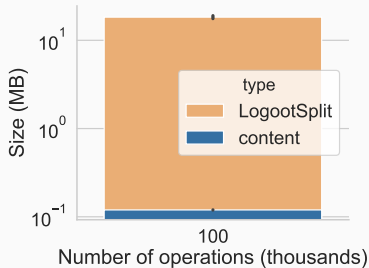
*position* allows to determine the position of this identifier compared to others

*node\_id* refers to the node's identifier, assumed to be unique

*node\_seq* refers to the node's logical clock, which increases monotonically with local operations

*offset* refers to the element position in its original block

Evergrowing overhead : impacts memory, bandwidth and CPU



Operation count : 100k

Size of content : 100KB

Size of data structure : 20MB

FIGURE 4 – Memory footprint of the data structure

How to reduce the overhead introduced by the data structure?



Core-nebula approach<sup>2</sup>

Reassigns shorter identifiers to elements...  
...but requires consensus

LSEQ<sup>3</sup>

Set of strategies to reduce the growth of identifiers ...  
...but overhead still proportional to number of elements

- 
2. Marek ZAWIRSKI et al. Asynchronous rebalancing of a replicated tree. In *Conférence Française en Systèmes d'Exploitation (CFSE)*, page 12, Saint-Malo, France, mai 2011. URL : <https://hal.inria.fr/hal-01248197>.
  3. Brice NÉDELEC et al. A scalable sequence encoding for collaborative editing. *Concurrency and Computation : Practice and Experience* :e4108. DOI : [10.1002/cpe.4108](https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4108). eprint : <https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4108>. URL : <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4108>.

## Our approach

Reassign shorter identifiers and aggregate them into blocks in a fully distributed manner

# RenamableLogootSplit

Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation

Can be performed without coordination



FIGURE 5 – Example of renaming

# RenamableLogootSplit

Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation

Can be performed without coordination

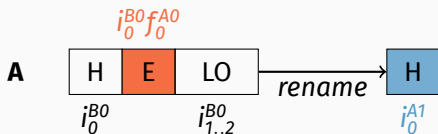


FIGURE 5 – Example of renaming

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

# RenamableLogootSplit

Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation

Can be performed without coordination

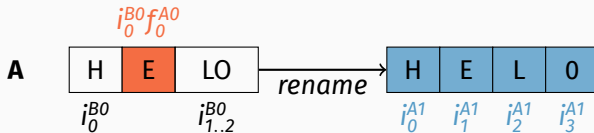


FIGURE 5 – Example of renaming

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

Then generates contiguous identifiers for all following elements

# RenamableLogootSplit

Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation

Can be performed without coordination

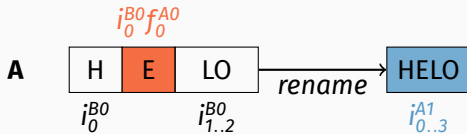


FIGURE 5 – Example of renaming

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

Then generates contiguous identifiers for all following elements

# Handling concurrent operations

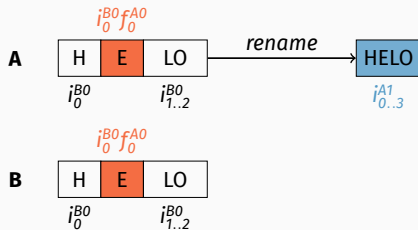


FIGURE 6 – Applying naively concurrent update

# Handling concurrent operations

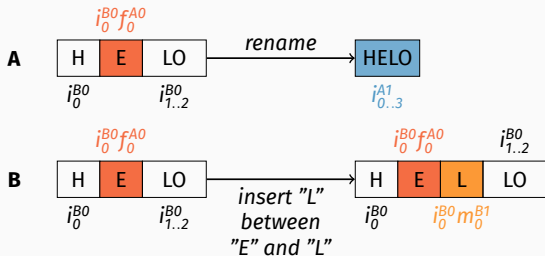


FIGURE 6 – Applying naively concurrent update

Can issue operations concurrently to *rename*



# Handling concurrent operations

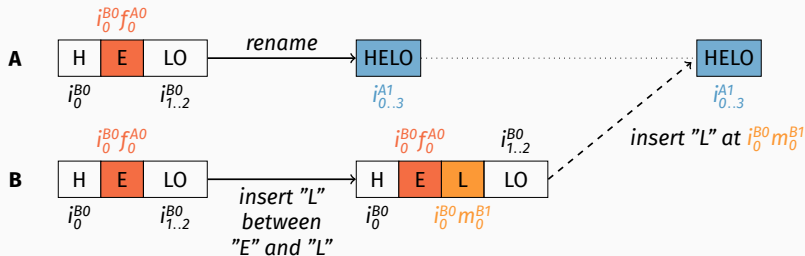


FIGURE 6 – Applying naively concurrent update

Can issue operations concurrently to *rename*

# Handling concurrent operations

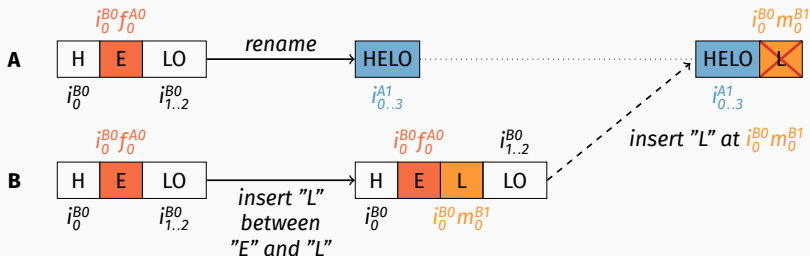


FIGURE 6 – Applying naively concurrent update

Can issue operations concurrently to *rename*  
Produce inconsistencies if applied naively

# Fixing handling concurrent operations

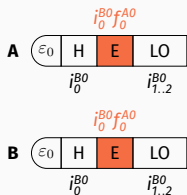


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

# Fixing handling concurrent operations

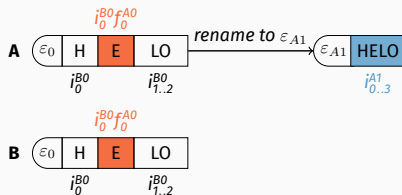


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

# Fixing handling concurrent operations

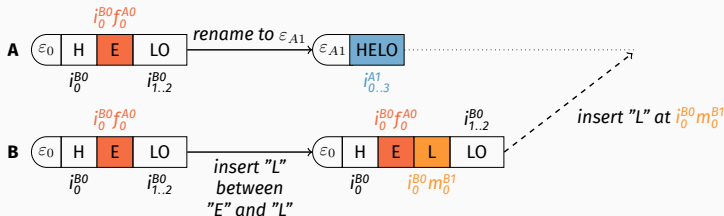


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

# Fixing handling concurrent operations

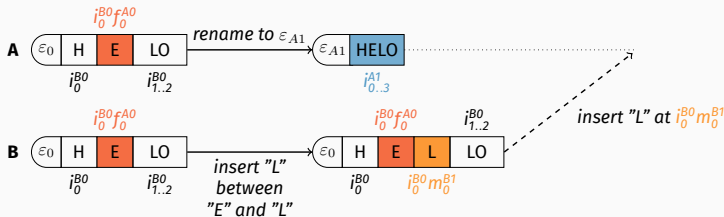


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

Transform operations against *rename* ones (OT)

# Fixing handling concurrent operations

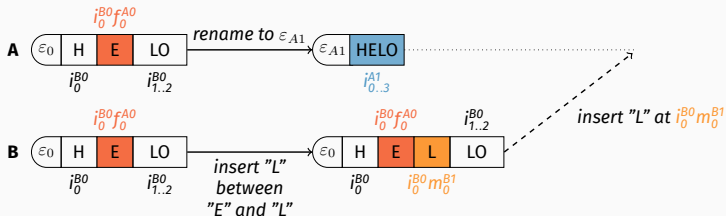


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

Transform operations against *rename* ones (OT)

(i) Find predecessor in former state ( $i_0^{B0} f_0^{A0}$ )

# Fixing handling concurrent operations

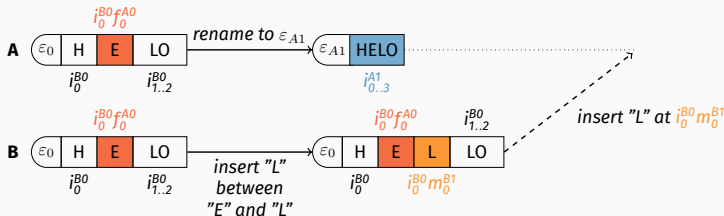


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

Transform operations against *rename* ones (OT)

- (i) Find predecessor in former state ( $i_0^{B0} f_0^{A0}$ )
- (ii) Find its counterpart in new state ( $i_1^{A1}$ )



# Fixing handling concurrent operations

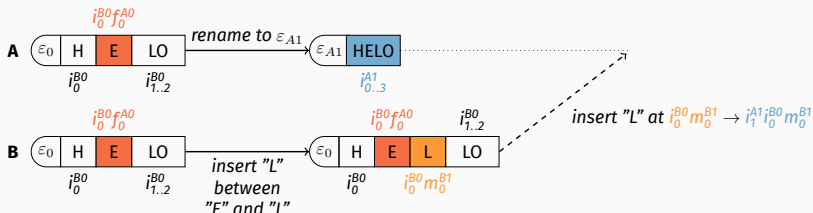


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

Transform operations against *rename* ones (OT)

- Find predecessor in former state ( $i_0^{B0}f_0^{A0}$ )
- Find its counterpart in new state ( $i_1^{A1}$ )
- Prepend it to given id to form new id ( $i_1^{A1}i_0^{B0}m_0^{B1}$ )

# Fixing handling concurrent operations

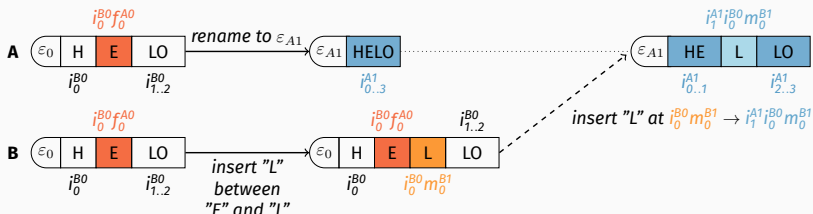


FIGURE 7 – Handling properly concurrent update

Use *epoch-based* system to track concurrent operations

Transform operations against *rename* ones (OT)

- Find predecessor in former state ( $i_0^{B0} f_0^{A0}$ )
- Find its counterpart in new state ( $i_1^{A1}$ )
- Prepend it to given id to form new id ( $i_1^{A1} i_0^{B0} m_0^{B1}$ )

What about concurrent *rename* operations?

# What about concurrent *rename* operations?

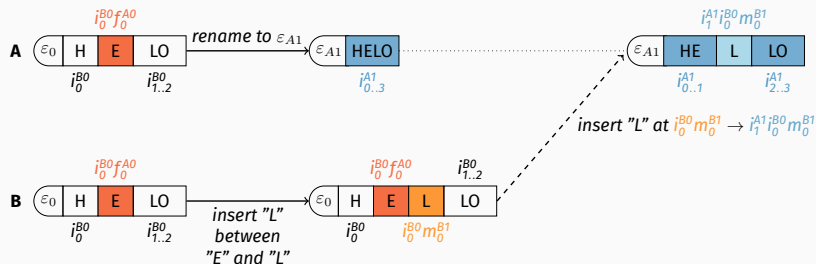


FIGURE 8 – Concurrent *rename* operations leading to divergent states

# What about concurrent *rename* operations?

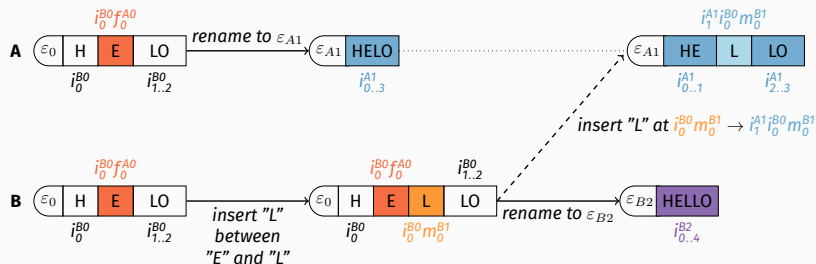


FIGURE 8 – Concurrent *rename* operations leading to divergent states

# What about concurrent *rename* operations?

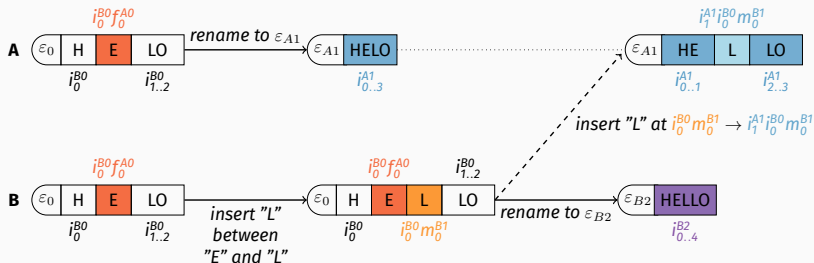


FIGURE 8 – Concurrent *rename* operations leading to divergent states

*Rename* operations are system operations

# What about concurrent *rename* operations?

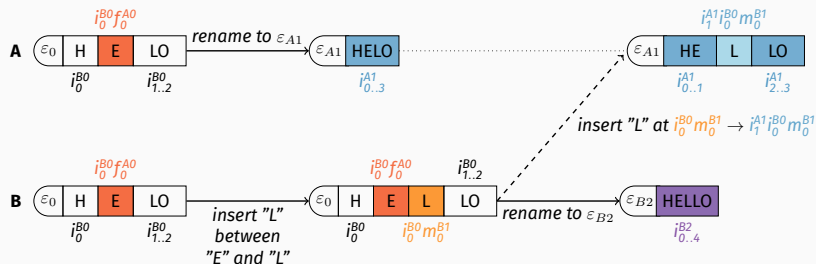


FIGURE 8 – Concurrent *rename* operations leading to divergent states

*Rename* operations are system operations

Can resolve conflict by only applying one of them

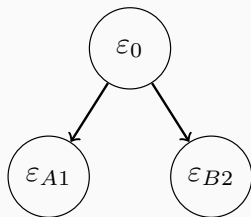


FIGURE 9 – *Epoch tree* corresponding to previous scenario



## How to do so?

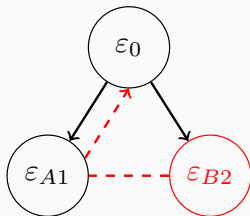


FIGURE 9 – *Epoch tree* corresponding to previous scenario

Have to pick an epoch as the **target one**

Define total order on epochs

## How to do so?

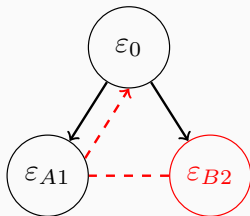


FIGURE 9 – Epoch tree corresponding to previous scenario

Have to pick an epoch as the **target one**

Define total order on epochs

Have to move through the tree

Design transformation function to revert *rename* operation

# Applying concurrent *rename* operations

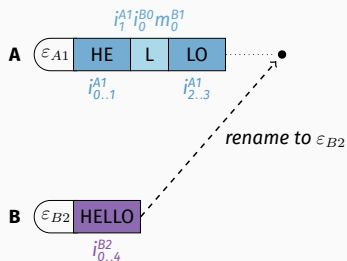


FIGURE 10 – Applying a concurrent *rename* operation

# Applying concurrent *rename* operations

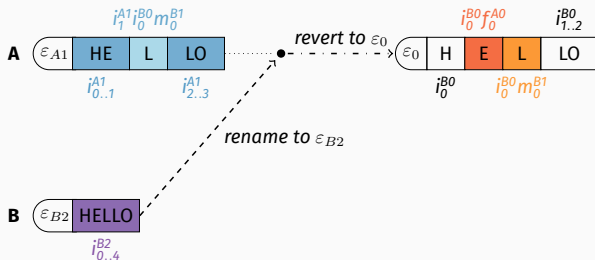


FIGURE 10 – Applying a concurrent *rename* operation

Revert state to equivalent one at LCA epoch

# Applying concurrent *rename* operations

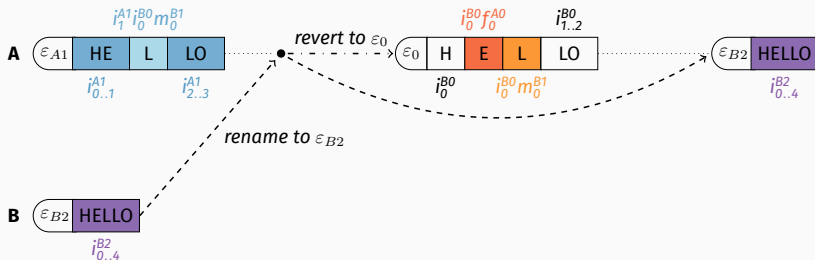


FIGURE 10 – Applying a concurrent *rename* operation

Revert state to equivalent one at LCA epoch

Apply then *rename* operations leading to target epoch

## Need to store former state until no more concurrent operations

Can garbage collect it once the *rename* operation is causally stable<sup>4</sup>

Can offload it to the disk meanwhile

## Need to propagate former state to other nodes

Can compress the operation to minimise bandwidth consumption

Can trigger *rename* operations at a given number of blocks

---

4. Carlos BAQUERO et al. Making operation-based crdts operation-based. In Kostas MAGOUTIS et al., éditeurs, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg. Springer Berlin Heidelberg, 2014 .

## Evaluation

Ran simulations to compare performance of  
RenamableLogootSplit to LogootSplit one

Simulate collaborative editing sessions using either LogootSplit or RenamableLogootSplit

**Phase 1 (content generation)** : 80/20% of *insert/remove*

**Phase 2 (editing)** : 50/50% of *insert/remove*

Nodes switch to phase 2 when document reaches critical size (15 pages - 60k elements)



# Experimental settings

Use Node.js version 13.1.0

Obtained documents sizes using our fork of *object-sizeof*<sup>5</sup>

Ran benchmarks on a workstation equipped of a Intel Xeon CPU E5-1620 (10MB Cache, 3.50 GHz) with 16GB of RAM running Fedora 31

Measured times using `process.hrtime.bigint()`

---

5. <https://www.npmjs.com/package/object-sizeof>

## Results - Convergence

Compared final content of nodes per sessions

Did not observe any divergence

Empirical result, not a proof...

... but represents first step towards the validation

# Results - Memory footprint

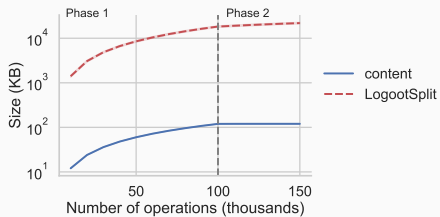


FIGURE 11 – Evolution of the size of the document

# Results - Memory footprint

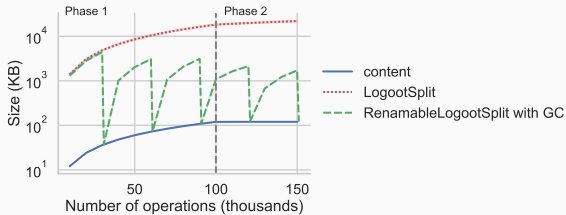


FIGURE 11 – Evolution of the size of the document

*Rename* resets the overhead of the CRDT, if can garbage collect

# Results - Memory footprint

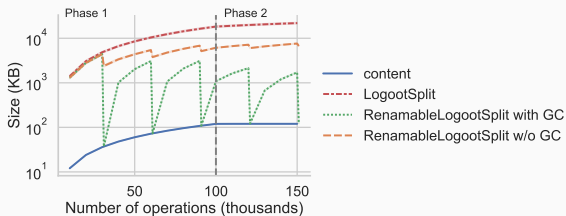
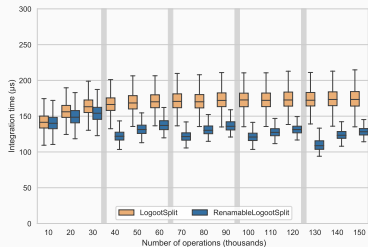


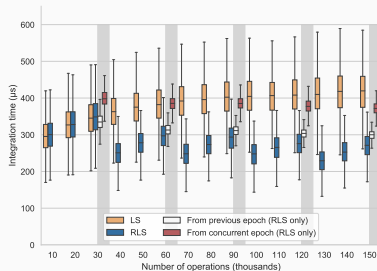
FIGURE 11 – Evolution of the size of the document

*Rename* resets the overhead of the CRDT, if can garbage collect  
*Rename* still reduces by 66% the size otherwise

# Results - Integration time of *insert* operations



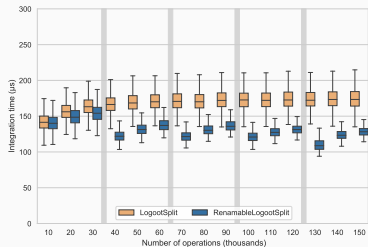
(a) Local operations



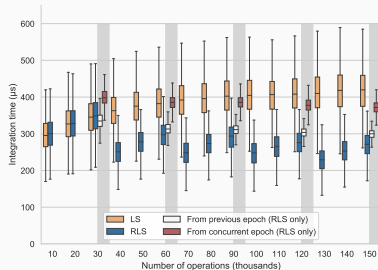
(b) Remote operations

FIGURE 12 – Evolution of the integration time of *insert* operations

# Results - Integration time of *insert* operations



(a) Local operations

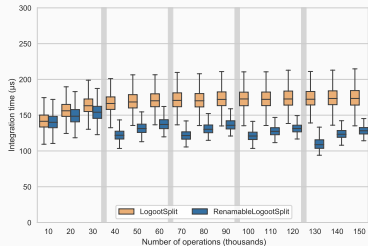


(b) Remote operations

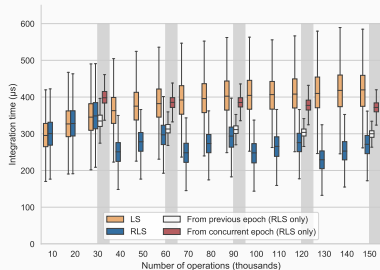
FIGURE 12 – Evolution of the integration time of *insert* operations

*Rename* reduces integration times of future operations

# Results - Integration time of *insert* operations



(a) Local operations



(b) Remote operations

FIGURE 12 – Evolution of the integration time of *insert* operations

*Rename* reduces integration times of future operations

Transforming concurrent operations is actually faster than applying them on former state



## Results - Integration time of *rename* operations

| Parameters |            | Integration Time (ms) |         |                         |       |
|------------|------------|-----------------------|---------|-------------------------|-------|
| Type       | Nb Ops (k) | Mean                  | Median  | 99 <sup>th</sup> Quant. | Std   |
| Local      | 30         | 41.75                 | 38.74   | 71.68                   | 6.84  |
|            | 90         | 119.19                | 118.87  | 124.22                  | 2.49  |
|            | 150        | 158.04                | 157.95  | 164.38                  | 2.49  |
| Remote     | 30         | 481.32                | 477.13  | 537.30                  | 17.11 |
|            | 90         | 1491.28               | 1481.83 | 1657.58                 | 51.10 |
|            | 150        | 1694.17               | 1675.95 | 1852.55                 | 59.94 |

**TABLE 1** – Integration time of rename operations

## Results - Integration time of *rename* operations

| Parameters |            | Integration Time (ms) |         |                         |       |
|------------|------------|-----------------------|---------|-------------------------|-------|
| Type       | Nb Ops (k) | Mean                  | Median  | 99 <sup>th</sup> Quant. | Std   |
| Local      | 30         | 41.75                 | 38.74   | 71.68                   | 6.84  |
|            | 90         | 119.19                | 118.87  | 124.22                  | 2.49  |
|            | 150        | 158.04                | 157.95  | 164.38                  | 2.49  |
| Remote     | 30         | 481.32                | 477.13  | 537.30                  | 17.11 |
|            | 90         | 1491.28               | 1481.83 | 1657.58                 | 51.10 |
|            | 150        | 1694.17               | 1675.95 | 1852.55                 | 59.94 |

**TABLE 1** – Integration time of rename operations

Noticeable by users

## Results - Integration time of *rename* operations

| Parameters |            | Integration Time (ms) |         |                         |       |
|------------|------------|-----------------------|---------|-------------------------|-------|
| Type       | Nb Ops (k) | Mean                  | Median  | 99 <sup>th</sup> Quant. | Std   |
| Local      | 30         | 41.75                 | 38.74   | 71.68                   | 6.84  |
|            | 90         | 119.19                | 118.87  | 124.22                  | 2.49  |
|            | 150        | 158.04                | 157.95  | 164.38                  | 2.49  |
| Remote     | 30         | 481.32                | 477.13  | 537.30                  | 17.11 |
|            | 90         | 1491.28               | 1481.83 | 1657.58                 | 51.10 |
|            | 150        | 1694.17               | 1675.95 | 1852.55                 | 59.94 |

**TABLE 1** – Integration time of rename operations

Noticeable by users

Need to improve remote integration time

## Results - Integration time of *rename* operations (complete)

| Parameters             |            | Integration Time (ms) |         |                         |       |
|------------------------|------------|-----------------------|---------|-------------------------|-------|
| Type                   | Nb Ops (k) | Mean                  | Median  | 99 <sup>th</sup> Quant. | Std   |
| Local                  | 30         | 41.75                 | 38.74   | 71.68                   | 6.84  |
|                        | 90         | 119.19                | 118.87  | 124.22                  | 2.49  |
|                        | 150        | 158.04                | 157.95  | 164.38                  | 2.49  |
| Direct remote          | 30         | 481.32                | 477.13  | 537.30                  | 17.11 |
|                        | 90         | 1491.28               | 1481.83 | 1657.58                 | 51.10 |
|                        | 150        | 1694.17               | 1675.95 | 1852.55                 | 59.94 |
| Cc. int. greater epoch | 30         | 643.53                | 643.57  | 682.80                  | 13.42 |
|                        | 90         | 1998.23               | 1994.08 | 2111.98                 | 45.37 |
|                        | 150        | 2241.92               | 2233.61 | 2351.02                 | 52.20 |
| Cc. int. lesser epoch  | 30         | 1.36                  | 1.30    | 3.53                    | 0.37  |
|                        | 90         | 4.45                  | 4.23    | 5.81                    | 0.71  |
|                        | 150        | 5.53                  | 5.26    | 8.70                    | 0.79  |

TABLE 2 – Integration time of rename operations

# Conclusion

## Done

Designed a new Sequence CRDT : RenamableLogootSplit

Validated it through experimental evaluation

# Conclusion

## Done

Designed a new Sequence CRDT : RenamableLogootSplit

Validated it through experimental evaluation

## Work in progress

Publishing it

Writing the manuscript

# Conclusion

## Done

- Designed a new Sequence CRDT : RenamableLogootSplit

- Validated it through experimental evaluation

## Work in progress

- Publishing it

- Writing the manuscript

## To do

- Prove formally the correctness of the mechanism

- Design better strategies to select the target epoch

- Improve performance of *rename* operations

Propose a strategy to avoid conflicting *rename* operations

How to minimise likelihood of concurrent *rename* operations without coordinating?

Propose a smarter strategy to choose the "winning" renaming

How to minimise the overall computations?



Thanks for your attention, any questions?

