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

April 21, 2020

Context

- Working on data replication in Local-first softwares[1]
- Currently focusing on *Sequence Conflict-free Replicated Data Types (CRDTs)*
    - Notably used for real-time collaborative text editing

[1] Martin Kleppmann et al. Local-first software: you own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, pages 154–178, Athens, Greece. Association for Computing Machinery, 2019. ISBN: 9781450369954. DOI: 10.1145/3359591.3359737. URL: https://doi.org/10.1145/3359591.3359737 .

LogootSplit[2]

- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here with the following formalism: $\mathit{position}^{\mathit{node\ id}\ \ \mathit{node\ seq}}_{\mathit{offset}}$

[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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, October 2013. DOI: 10.4108/icst.collaboratecom.2013.254123 .

LogootSplit[2]

- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here with the following formalism: $position_{offset}^{node\ id\ node\ seq}$

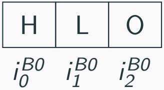
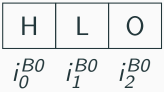


Figure 1: State of a sequence which contains the elements "hlo" and their corresponding identifiers

[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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, October 2013. DOI: 10.4108/icst.collaboratecom.2013.254123 .

LogootSplit[2]

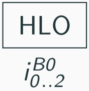
- State of the art of *Sequence CRDTs*
- Elements are ordered by their identifier, noted here with the following formalism: $position_{offset}^{node\ id\ node\ seq}$



H L O

i_0^{B0} i_1^{B0} i_2^{B0}

Figure 1: State of a sequence which contains the elements "hlo" and their corresponding identifiers



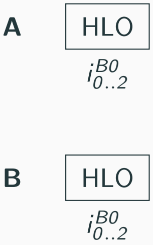
HLO

$i_{0..2}^{B0}$

Figure 2: State of a sequence which contains the block "hlo"

[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*, pages 50–59, Austin, TX, USA. IEEE Computer Society, October 2013. DOI: 10.4108/icst.collaboratecom.2013.254123 .

Example



A HLO $i^{B0}_{0..2}$

B HLO $i^{B0}_{0..2}$

Figure 3: Example of concurrent *insert* operations

Example

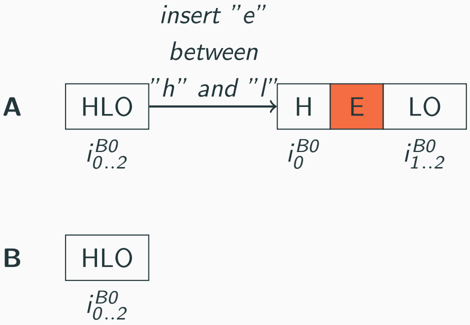


Figure 3: Example of concurrent *insert* operations

Example

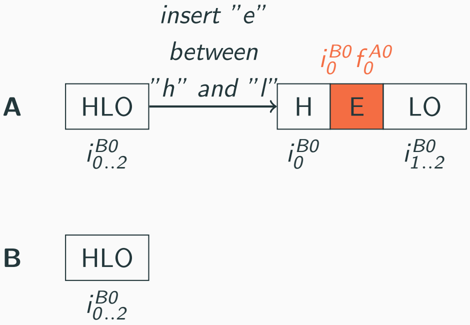


Figure 3: Example of concurrent *insert* operations

Example

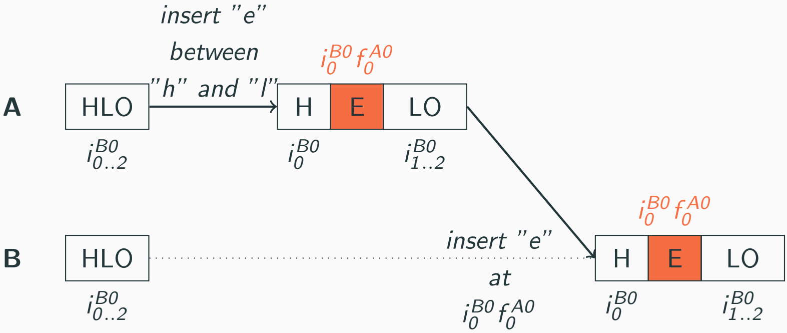


Figure 3: Example of concurrent *insert* operations

Research issue

Limits

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

Research issue

Limits

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

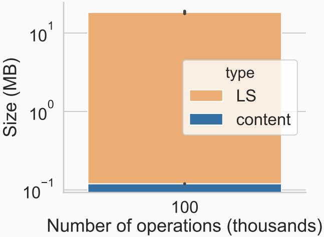


Figure 4: Footprint of the data structure

How to reduce the overhead introduced by the data structure ?

Related work

- Core-nebula approach[3]
  - Reassigns shorter identifiers to elements. . .
  - . . . but requires consensus
- LSEQ[4]
  - Set of strategies to reduce the growth of identifiers . . .
  - . . . but overhead still proportional to number of elements

[3] Marek Zawirski et al. Asynchronous rebalancing of a replicated tree. In *Conférence Française en Systèmes d'Exploitation (CFSE)*, page 12, Saint-Malo, France, May 2011. URL: `https://hal.inria.fr/hal-01248197` .

[4] Brice Nédelec et al. A scalable sequence encoding for collaborative editing. *Concurrency and Computation: Practice and Experience*:e4108. 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 perform without coordination
- Today, focus on scenario without concurrent *rename* operations

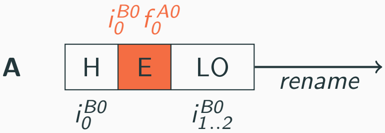


Figure 5: Example of renaming

RenamableLogootSplit

- Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation
- Can be perform without coordination
- Today, focus on scenario without concurrent *rename* operations

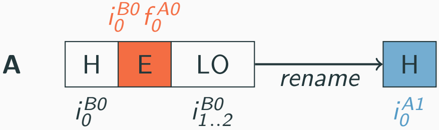


Figure 5: Example of renaming

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

RenamableLogootSplit

- Propose *RenamableLogootSplit*, *LogootSplit* with a *rename* operation
- Can be perform without coordination
- Today, focus on scenario without concurrent *rename* operations

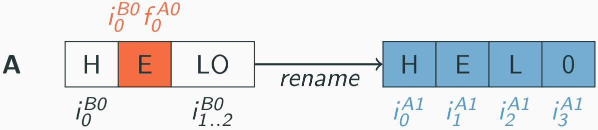


Figure 5: Example of renaming

- 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 perform without coordination
- Today, focus on scenario without concurrent *rename* operations

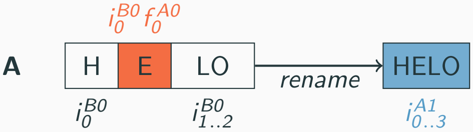


Figure 5: Example of renaming

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

Handling concurrent operations - part 1

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

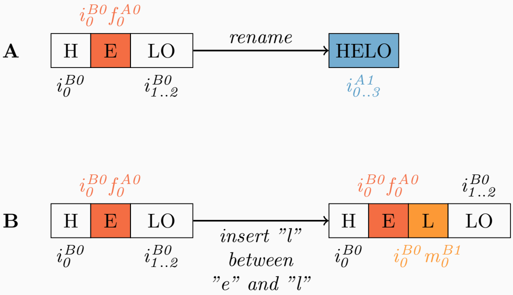


Figure 6: Example of concurrent insert

Handling concurrent operations - part 1

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

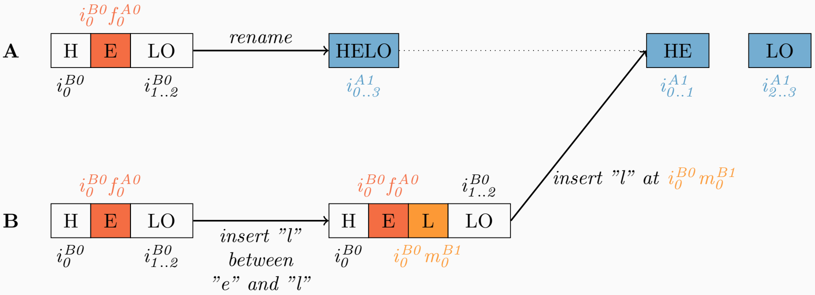


Figure 6: Example of concurrent insert

Handling concurrent operations - part 1

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

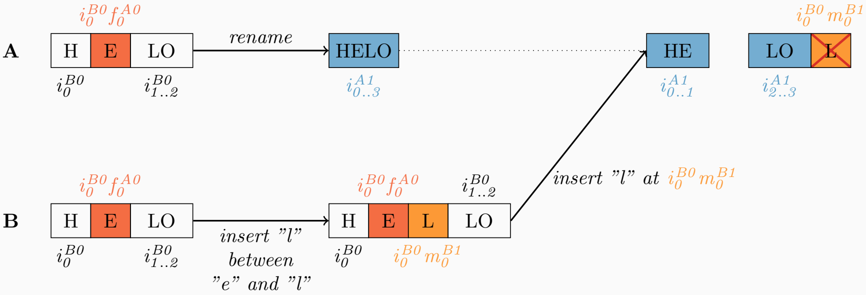


Figure 6: Example of concurrent insert

- May lead to inconsistencies

Handling concurrent operations - part 2

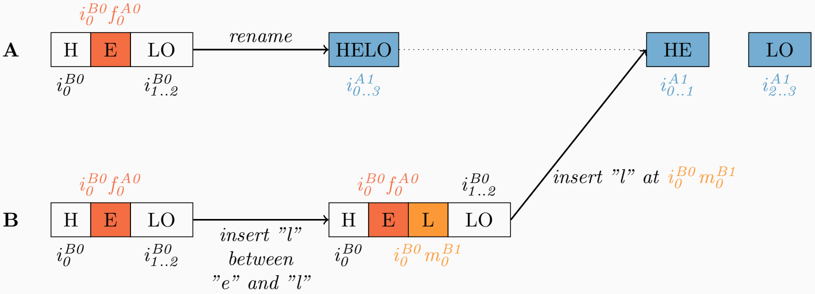


Figure 7: Example of concurrent insert

Handling concurrent operations - part 2

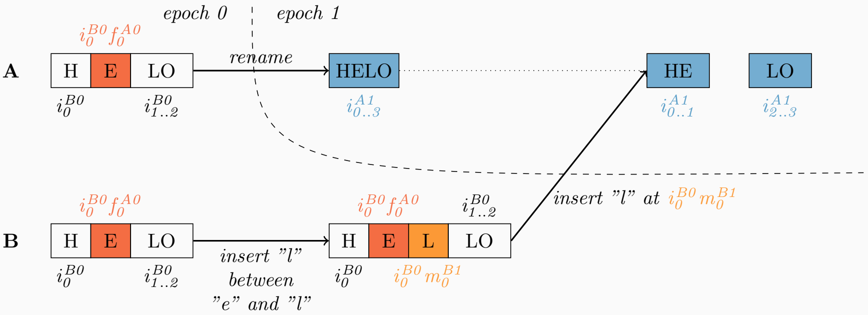


Figure 7: Example of concurrent insert

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

Handling concurrent operations - part 2

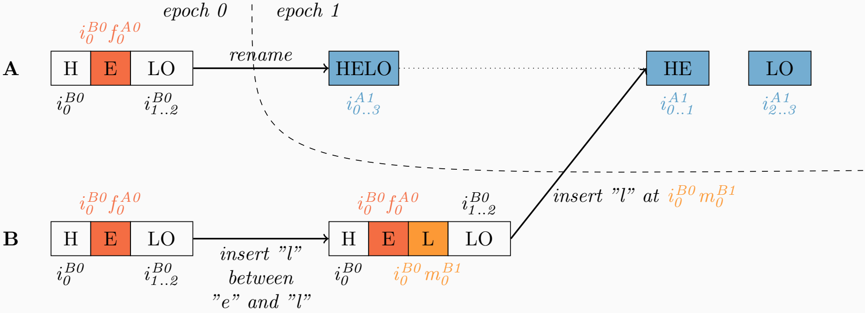


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations
- Transform operations against *rename* ones (*OT*)

Handling concurrent operations - part 2

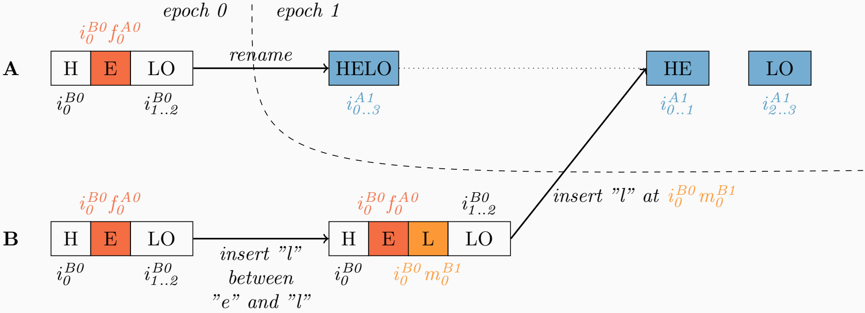


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations
- Transform operations against *rename* ones (*OT*)
  1. Find predecessor in former state ($i_0^{B0} f_0^{A0}$)

Handling concurrent operations - part 2

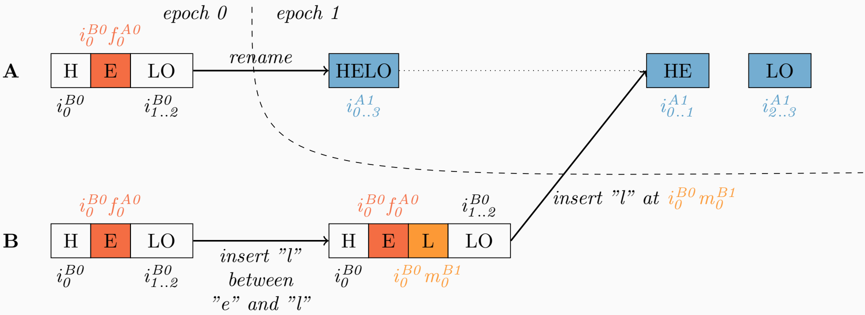


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations
- Transform operations against *rename* ones (*OT*)
  1. Find predecessor in former state ($i_0^{B0} f_0^{A0}$)
  2. Find its counterpart in new state (i_1^{A1})

Handling concurrent operations - part 2

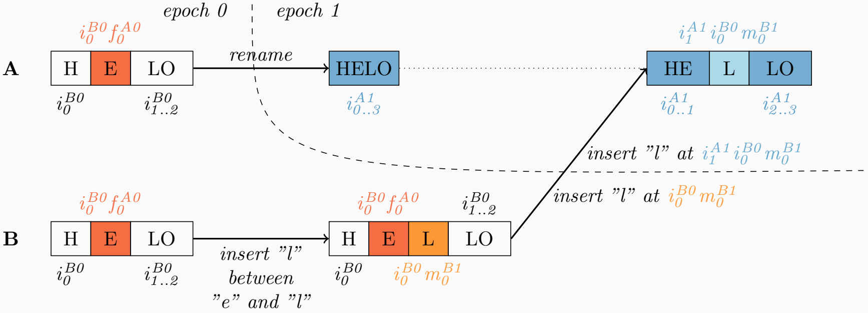


Figure 7: Example of concurrent insert

- Use *epoch-based* system to track concurrent operations
- Transform operations against *rename* ones (*OT*)
  1. Find predecessor in former state ($i_0^{B0} f_0^{A0}$)
  2. Find its counterpart in new state (i_1^{A1})
  3. Prepend it to given id to form new id ($i_1^{A1} i_0^{B0} f_0^{A0}$)

Downsides

Need to store former state until no more concurrent operations

[5] Carlos Baquero et al. Making operation-based crdts operation-based. In Kostas Magoutis et al., editors, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg. Springer Berlin Heidelberg, 2014 .

Downsides

Need to store former state until no more concurrent operations

- Can garbage collect it once the *rename* operation is causally stable[5]
- Can offload it to the disk meanwhile

[5] Carlos Baquero et al. Making operation-based crdts operation-based. In Kostas Magoutis et al., editors, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg. Springer Berlin Heidelberg, 2014 .

Downsides

Need to store former state until no more concurrent operations

- Can garbage collect it once the *rename* operation is causally stable[5]
- Can offload it to the disk meanwhile

Need to propagate former state to other nodes

[5] Carlos Baquero et al. Making operation-based crdts operation-based. In Kostas Magoutis et al., editors, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg. Springer Berlin Heidelberg, 2014 .

Downsides

Need to store former state until no more concurrent operations

- Can garbage collect it once the *rename* operation is causally stable[5]
- Can offload it to the disk meanwhile

Need to propagate former state to other nodes

- Can compress the operation to minimise bandwidth consumption

[5] Carlos Baquero et al. Making operation-based crdts operation-based. In Kostas Magoutis et al., editors, *Distributed Applications and Interoperable Systems*, pages 126–140, Berlin, Heidelberg. Springer Berlin Heidelberg, 2014 .

Evaluation

Scenario

- **Assumption:** Only one node can issue *rename* operations

Ran simulations to evaluate proposed approach:

- 10 nodes share and edit a document collaboratively
- Nodes use either LogootSplit or RenamableLogootSplit according to session

Scenario

- **Assumption:** Only one node can issue *rename* operations

Ran simulations to evaluate proposed approach:

- 10 nodes share and edit a document collaboratively
- Nodes use either LogootSplit or RenamableLogootSplit according to session
- **Phase 1 (content generation):** 80/20% of *insert*/*remove*
- **Phase 2 (refactoring):** 50/50% of *insert*/*remove*
- Nodes switch to phase 2 when document reaches critical size (15 pages - 60k elements)

Scenario

- **Assumption:** Only one node can issue *rename* operations

Ran simulations to evaluate proposed approach:

- 10 nodes share and edit a document collaboratively
- Nodes use either LogootSplit or RenamableLogootSplit according to session
- **Phase 1 (content generation):** 80/20% of *insert*/*remove*
- **Phase 2 (refactoring):** 50/50% of *insert*/*remove*
- Nodes switch to phase 2 when document reaches critical size (15 pages - 60k elements)
- Overall, nodes perform 150k operations on the document
- In the case of *RenamableLogootSplit*, trigger a *rename* operation every 30k operations

Results - Convergence

- Compared final content of nodes per sessions
- Did not observe any divergence

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 - Overhead of the data structure

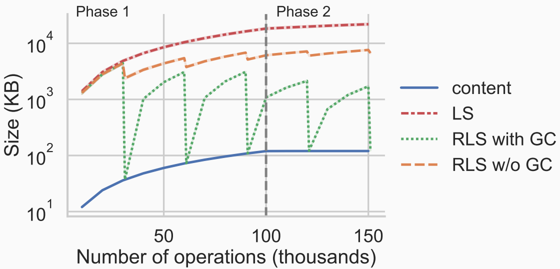


Figure 8: Evolution of the size of the document

Results - Overhead of the data structure

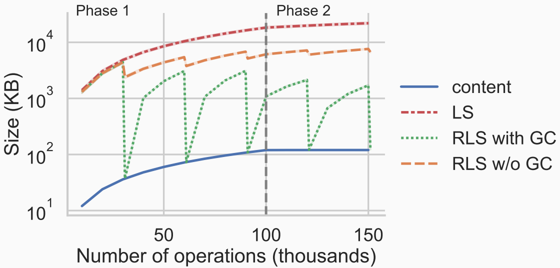


Figure 8: Evolution of the size of the document

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

Results - Overhead of the data structure

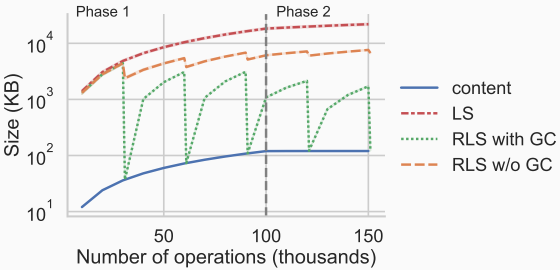
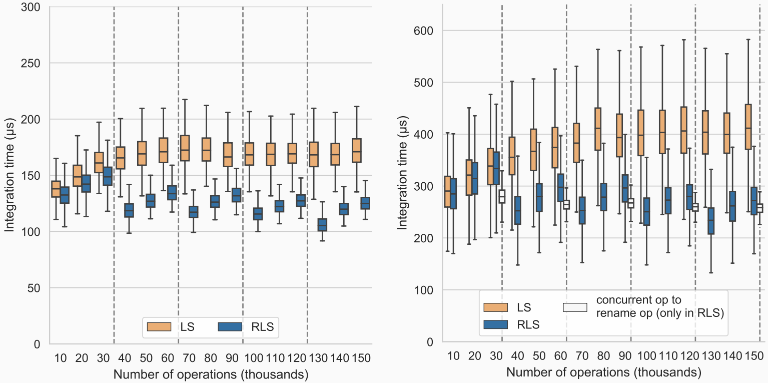


Figure 8: Evolution of the size of the document

- *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

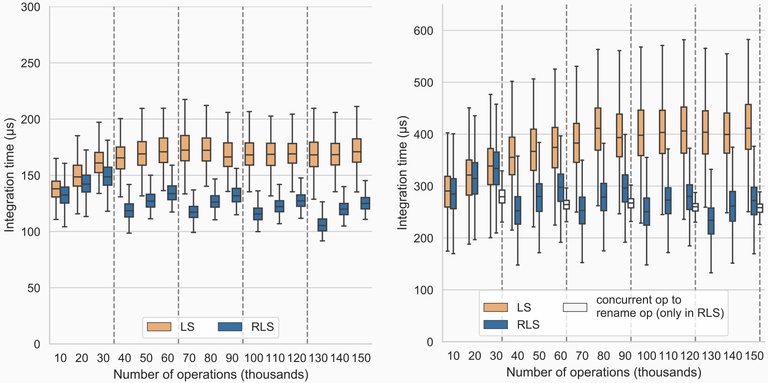


(a) Local operations

(b) Remote operations

Figure 9: Evolution of the integration time of *insert* operations

Results - Integration time of insert operations



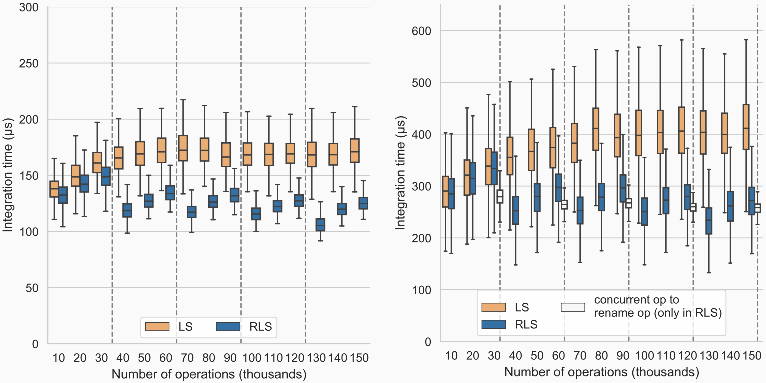
(a) Local operations

(b) Remote operations

Figure 9: Evolution of the integration time of *insert* operations

- *Rename* resets integration times of future operations

Results - Integration time of insert operations



(a) Local operations

(b) Remote operations

Figure 9: Evolution of the integration time of *insert* operations

- *Rename* resets integration times of future operations
- Transforming concurrent operations is actually faster than applying them on former state

Results - Integration time of rename operations

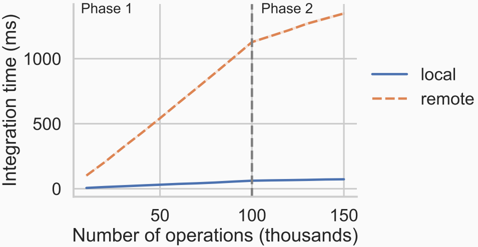


Figure 10: Evolution of the integration time of *rename* operations

Results - Integration time of rename operations

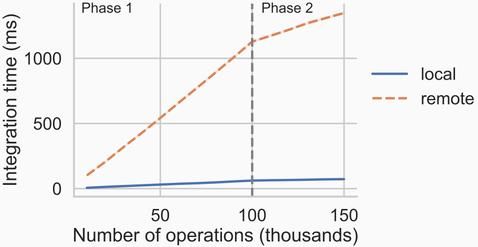


Figure 10: Evolution of the integration time of *rename* operations

- Noticeable by users if delayed too much

Results - Integration time of rename operations

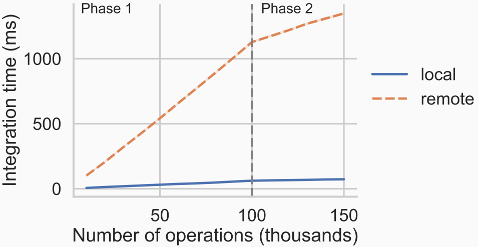


Figure 10: Evolution of the integration time of *rename* operations

- Noticeable by users if delayed too much
- When to trigger *rename* operations?

To wrap up

Done

- Designed a *rename* operation for LogootSplit
- Defined transformation functions to deal with concurrent updates

[6] 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 transformation functions to deal with concurrent updates

Work in progress

- Implementing support of concurrent *rename* operations in MUTE[6], our P2P collaborative text editor
- Proceeding to its validation
- Designing the strategy to trigger *rename* operations

[6] 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 transformation functions to deal with concurrent updates

Work in progress

- Implementing support of concurrent *rename* operations in MUTE[6], our P2P collaborative text editor
- Proceeding to its validation
- Designing the strategy to trigger *rename* operations

To do

- Prove formally the correctness of the mechanism
- Investigate the combination of OT techniques and CRDTs

[6] 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 .

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

Handling concurrent rename

The topic of a later contribution

Handling concurrent rename

The topic of a later contribution

rename operation not commutative

Handling concurrent rename

The topic of a later contribution

rename operation not commutative

To fix this:

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

Perspectives

Propose a strategy to avoid conflicting rename operations

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

Perspectives

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?

Experimental settings

- Use Node.js version 13.1.0
- Obtained documents sizes using our fork of *object-sizeof* [7]
- 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()`

[7] `https://www.npmjs.com/package/object-sizeof`