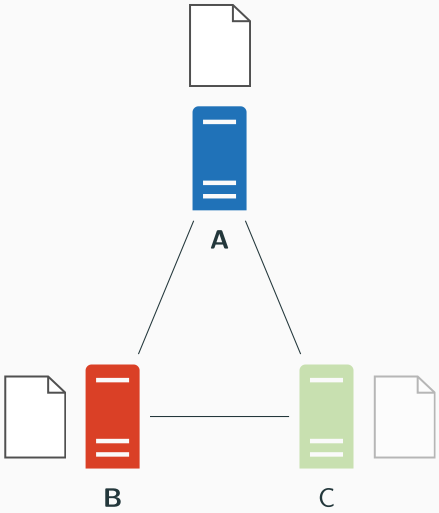
Efficient renaming in Conflict-free Replicated Data Types (CRDTs)

Case Study of a Sequence CRDT : LogootSplit

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

January 9, 2020

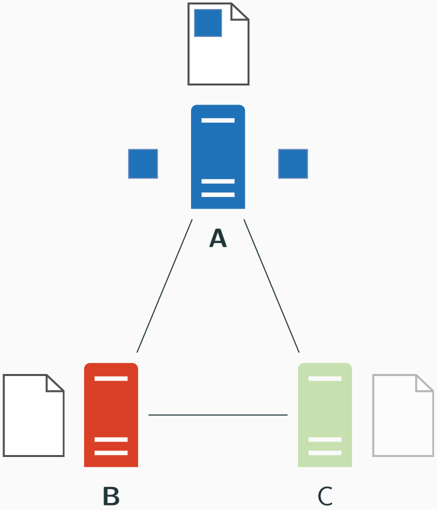
Conflict-free Replicated Data Types (CRDTs)[1]



- 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 .

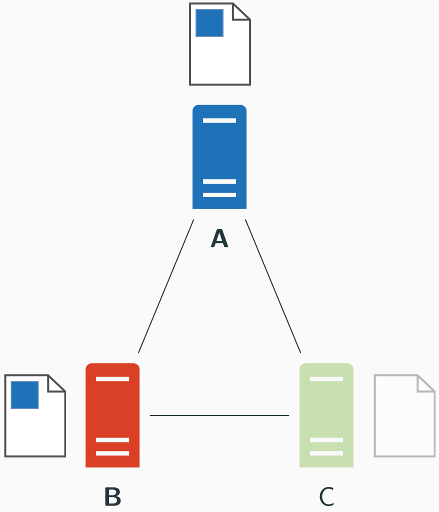
Conflict-free Replicated Data Types (CRDTs)[1]



- 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 .

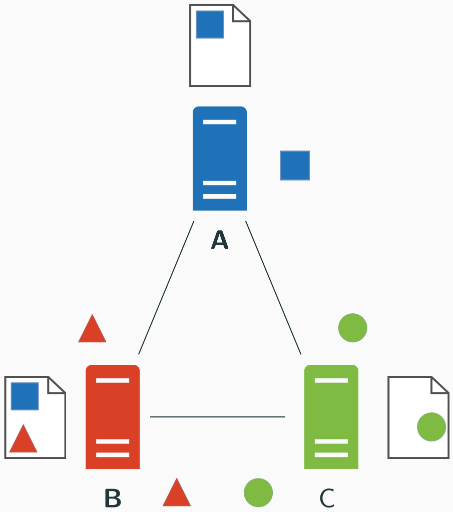
Conflict-free Replicated Data Types (CRDTs)[1]



- 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 .

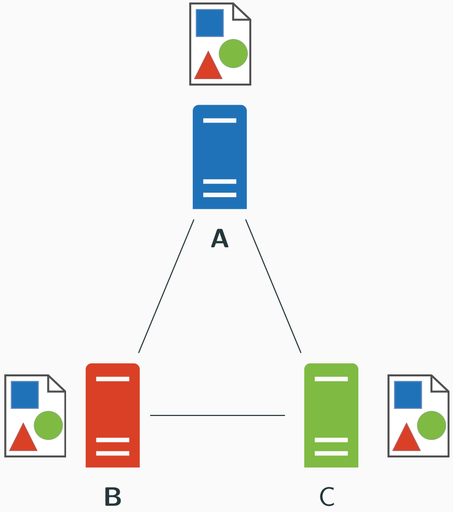
Conflict-free Replicated Data Types (CRDTs)[1]



- 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 concurrent updates
- ...

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

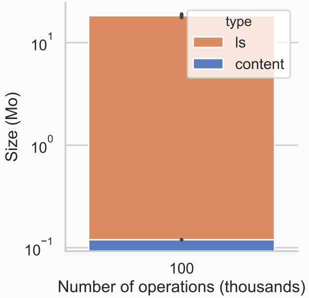


Figure 1: Footprint of the data structure

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

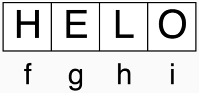


Figure 2: State of a sequence which contains the elements "helo" and their corresponding identifiers

[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

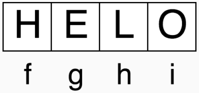


Figure 2: State of a sequence which contains the elements "helo" and their corresponding identifiers

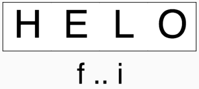


Figure 3: State of a sequence which contains the block "helo"

[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

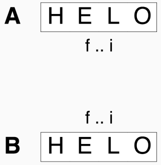


Figure 4: Example of concurrent *insert* operations

Example

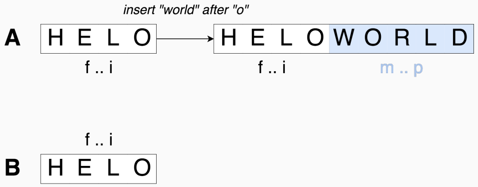


Figure 4: Example of concurrent *insert* operations

Example

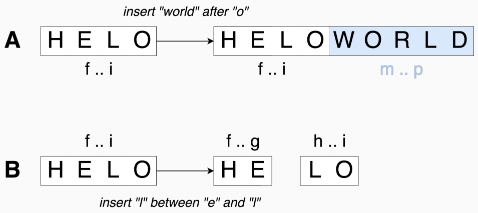


Figure 4: Example of concurrent *insert* operations

Example

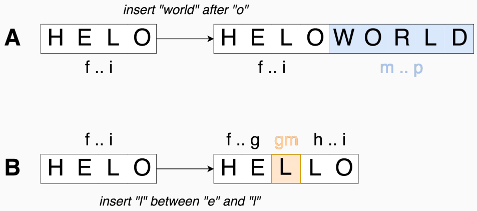


Figure 4: Example of concurrent *insert* operations

Example

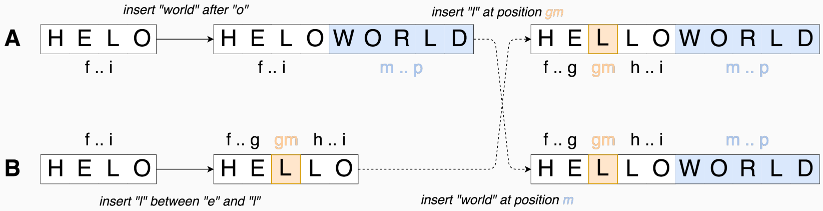


Figure 4: Example of concurrent *insert* operations

Declining performances

- Updates performed may lead to an inefficient internal representation

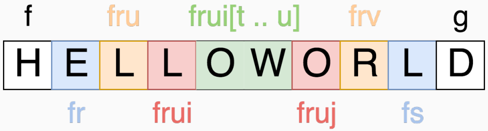


Figure 5: Example of inefficient internal representation

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

Our approach

RenamableLogootSplit

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

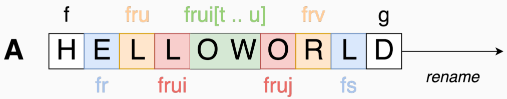


Figure 6: Example of renaming

RenamableLogootSplit

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

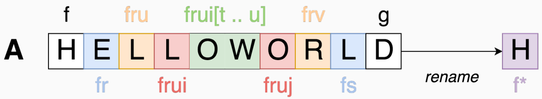


Figure 6: Example of renaming

- 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

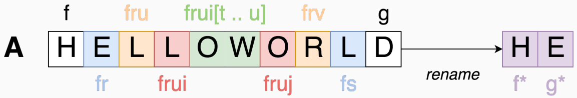


Figure 6: Example of renaming

- 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

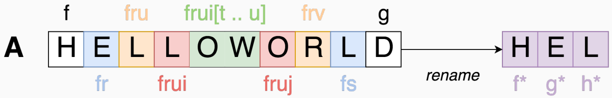


Figure 6: Example of renaming

- 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

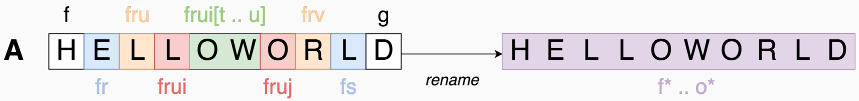


Figure 6: Example of renaming

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

Handling concurrent operations

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

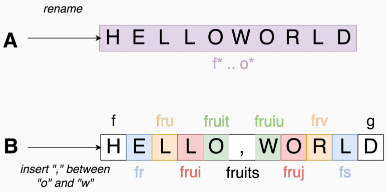


Figure 7: Example of concurrent insert

Handling concurrent operations

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

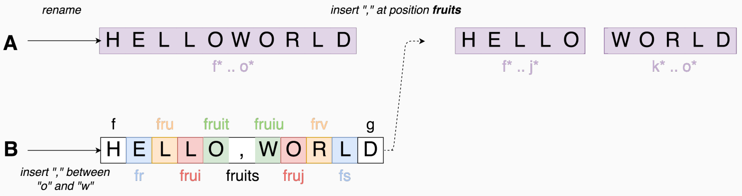


Figure 7: Example of concurrent insert

Handling concurrent operations

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

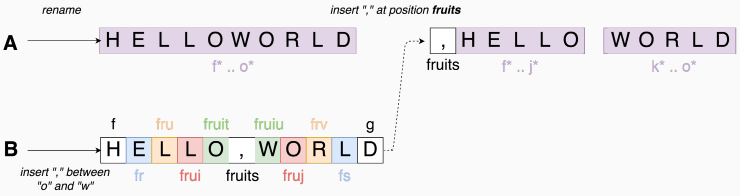


Figure 7: Example of concurrent insert

Handling concurrent operations

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

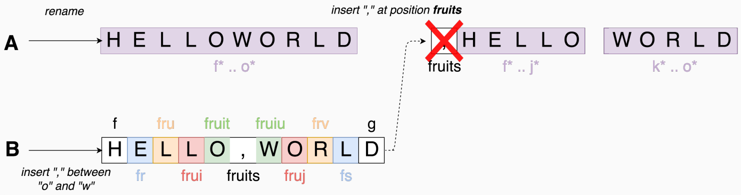


Figure 7: Example of concurrent insert

Handling concurrent operations

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

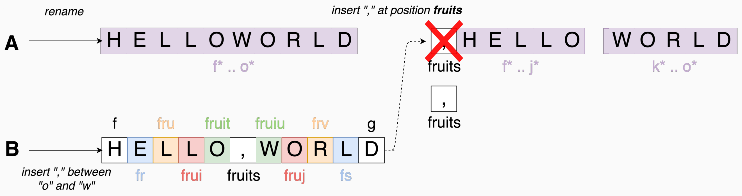


Figure 7: Example of concurrent insert

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

Handling concurrent operations

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

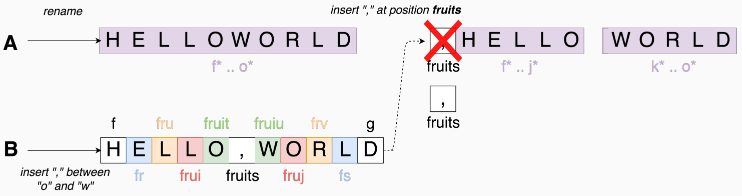


Figure 7: Example of concurrent insert

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

Handling concurrent operations

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

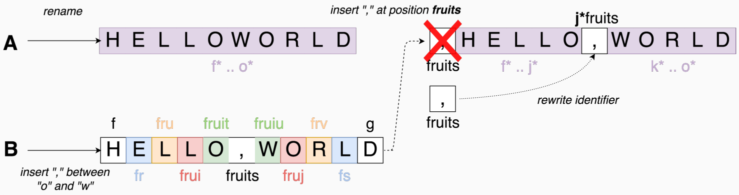


Figure 7: Example of concurrent insert

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

Downsides

Need to store former state until no more concurrent operations

[3]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[3]
- Can offload it to the disk meanwhile

[3]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[3]
- Can offload it to the disk meanwhile

Need to propagate former state to other nodes

[3]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[3]
- Can offload it to the disk meanwhile

Need to propagate former state to other nodes

- Can compress the operation to minimize bandwidth consumption

[3]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

Experimental settings

- Use Node.js version 13.1.0
- Obtained documents sizes using our fork of *object-sizeof* [4]
- 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()`

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

Results - Overhead of the data structure

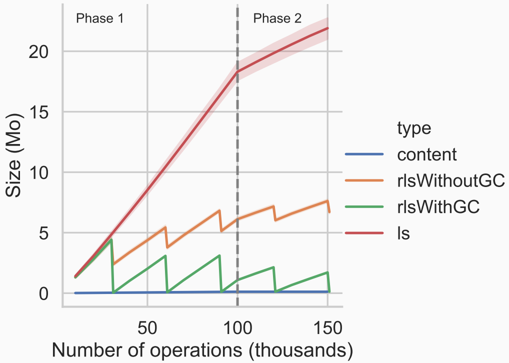


Figure 8: Evolution of the size of the document

Results - Overhead of the data structure

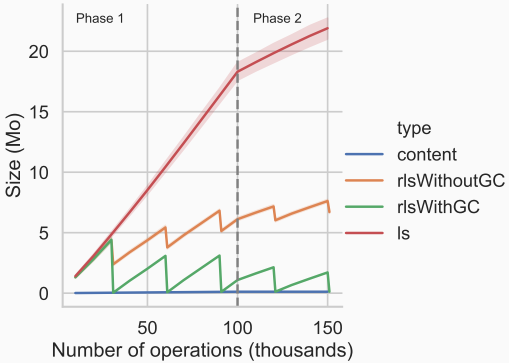


Figure 8: Evolution of the size of the document

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

Results - Overhead of the data structure

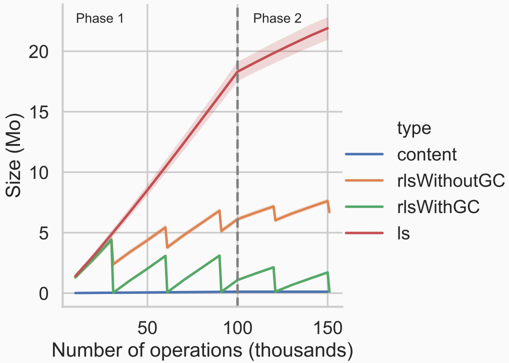
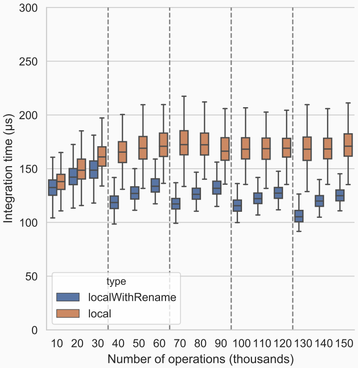


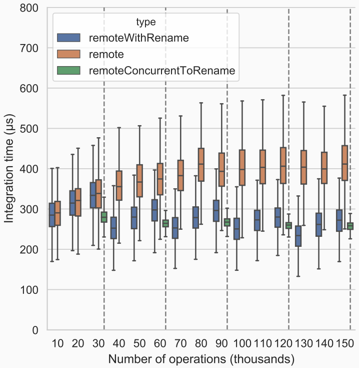
Figure 8: Evolution of the size of the document

- *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



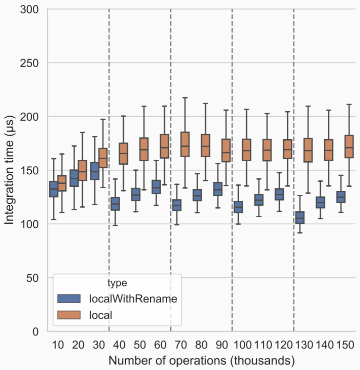
(a) Local operations



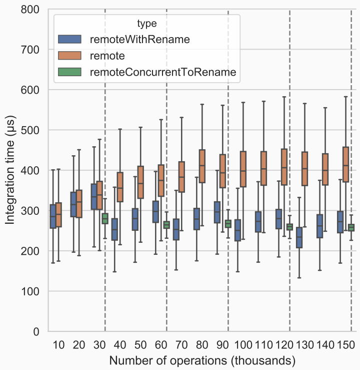
(b) Remote operations

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

Results - Integration time of insert operations



(a) Local operations

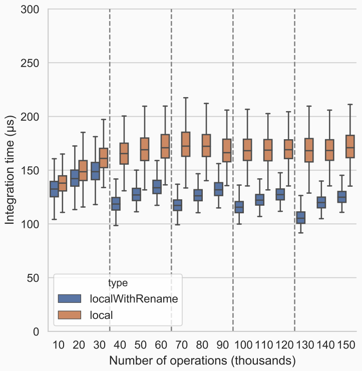


(b) Remote operations

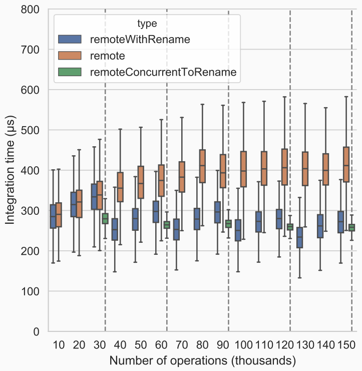
Figure 9: Evolution of the integration time of *insert* operations

- *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

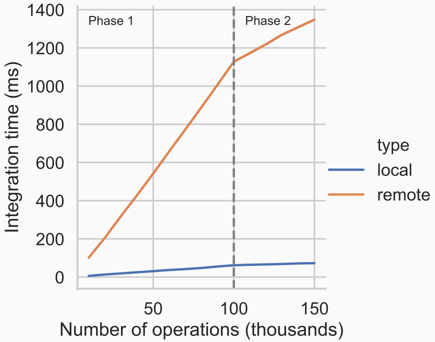


Figure 10: Evolution of the integration time of *rename* operations

Results - Integration time of rename operations

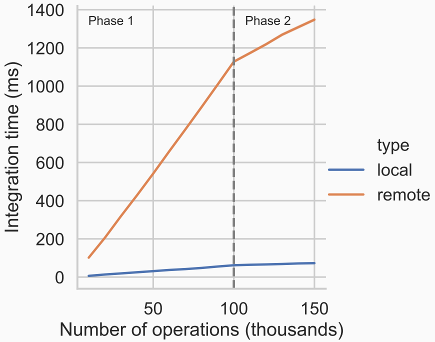


Figure 10: Evolution of the integration time of *rename* operations

- Noticeable by users if delayed too much

Results - Integration time of rename operations

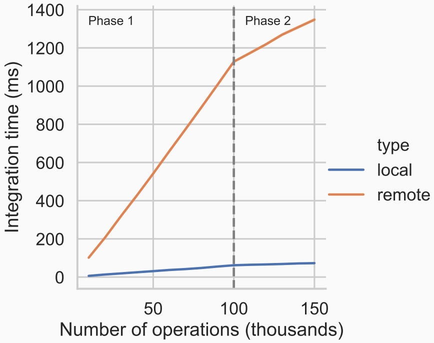


Figure 10: Evolution of the integration time of *rename* operations

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

To wrap up

Done

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

[5]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[5], our P2P collaborative text editor
- Benchmarking its performances
- Designing the strategy to trigger *rename* operations

[5]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[5], our P2P collaborative text editor
- Benchmarking its performances
- Designing the strategy to trigger *rename* operations

To do

- Publish it
- Prove formally the correctness of the mechanism

[5]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?

Perspectives

Propose a strategy to avoid conflicting rename operations

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

Perspectives

Propose a strategy to avoid conflicting rename operations

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

Propose a smarter strategy to choose the "winning" renaming

- How to minimize the overall computations?

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 rewriting rules to *undo* the effect of "losing" ones