

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

Matthieu Nicolas (matthieu.nicolas@loria.fr)

20 décembre 2022

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

TODO : Voir comment représenter une appli collaboratrice à ce stade

- Un **système collaboratif** est un système supportant ses utilisateur-rices dans leurs processus de collaboration pour la réalisation de tâches.

TODO : Voir comment illustrer ce point. Screenshots et nombre d'utilisateurs en-dessous ?

Avantages d'une architecture basée sur le cloud...

TODO : Représenter collaboration via appli basée sur le cloud

- **Disponibilité** : Répond aux utilisateur-rices
- **Tolérance aux pannes** : Fonctionne malgré pannes
- **Capacité de passage à l'échelle** : Supporte activité massive

TODO : Illustrer chacune des propriétés

- Confidentialité
- Pérennité
- Souveraineté
- Résistance à la censure

Pouvons-nous concevoir des applications collaboratives satisfaisant l'ensemble de ces propriétés?

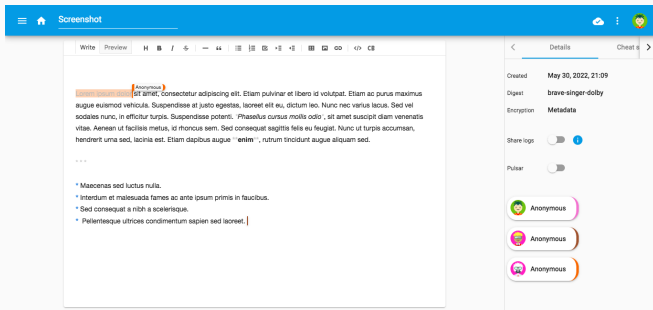
TODO : Illustrer une appli P2P

Problématiques

En l'absence d'autorités centrales, comment

- résoudre les conflits de modifications ?
- authentifier les utilisateur-rices ?
- sécuriser les communications ?

[1]. KLEPPMANN et al., « Local-First Software : You Own Your Data, in Spite of the Cloud ».



- Éditeur de texte collaboratif P2P temps réel chiffré de bout en bout
- Permet à l'équipe d'étudier et contribuer sur les problématiques des applications Local-First Software (LFS)

*. Disponible à : <https://mutehost.loria.fr>

TODO : À retravailler pour faire apparaître les problématiques ?

TODO : Ajouter comparaison des modèles de sync et approches pour CRDTs pour le type Séquence ?

- Conception d'un nouveau Conflict-free Replicated Data Type (CRDT) pour le type Séquence
- Implémentation et intégration de CRDTs dans MUTE

TODO : Représenter une séquence, puis une exécution provoquant un conflit

- Représente une collection ordonnée et de taille dynamique d'éléments
- Deux opérations de modifications, *insert* et *remove*
- Modifications ne commutent généralement pas

Un mécanisme de résolution de conflits est nécessaire pour résoudre les divergences

Conflict-free Replicated Data Types (CRDTs)^[2]

- Nouvelles spécifications des types de données
- Types de données répliquées
 - Permettent modifications sans coordination
 - Garantissent la convergence forte

Convergence forte

Ensemble des noeuds ayant intégrés le même ensemble de modifications obtient des états équivalents, sans nécessiter d'actions ou messages supplémentaires

[2]. SHAPIRO et al., « Conflict-Free Replicated Data Types ».

- Reposent sur la théorie des treillis

TODO : Représenter un sup-demi-treillis

- États ordonnés par relation \leq , avec état minimal \perp
- Modification d'un état génère état supérieur ou égal d'après \leq
- Existe fonction qui fusionne deux états et retourne leur borne supérieure

[3]. DAVEY et al., *Introduction to Lattices and Order*.

La fonction de fusion d'états est un mécanisme
de résolution de conflits automatique

Caractéristiques d'un CRDT

Ce qui définit un CRDT

- Sémantique de son mécanisme de résolution de conflits
- Modèle de synchronisation utilisé

Se focalise sur

- Spécification forte des séquences répliquées avec entrelacements^[4]
- Synchronisation par opérations

[4]. ATTIYA et al., « Specification and space complexity of collaborative text editing ».

- CRDT pour le type Séquence
- Appartient à l'approche à identifiants densément ordonnés

[5]. ANDRÉ et al., « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ».

Approche à identifiants densément ordonnés

- Assigne un identifiant de position à chaque élément de la séquence
- Utilise les identifiants des éléments pour les ordonner les uns par rapport aux autres

Propriétés des identifiants de position^[6]

- Unique
- Immuable
- Ordonnable par une relation d'ordre strict total $<_{id}$
- Appartenant à un espace dense

[6]. PREGUICA et al., « A Commutative Replicated Data Type for Cooperative Editing ».

- Composé d'un ou plusieurs tuples $pos_{offset}^{nodeId \ nodeSeq}$
- TODO : Ajouter animations pour expliciter le rôle de chaque composant

Exemple :

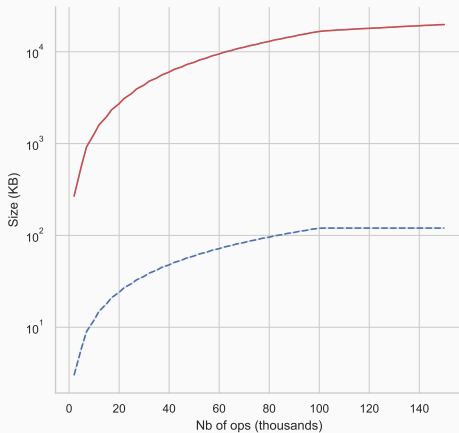
$$i_0^{A1} r_3^{B1} \dots m_0^{A2}$$

TODO : Reprendre exemple de divergence, mais avec LogootSplit pour montrer convergence

Limites de LogootSplit

- Croissance non-bornée de la taille des identifiants
- Fragmentation en blocs courts

Taille du contenu comparé à la taille de la séquence LogootSplit



1% contenu, 99%
métadonnées

L'approche core-nebula^[7]

- Ré-assigne des identifiants courts aux éléments, c.-à-d. les *renomme*
- Nécessite de transformer les opérations *insert* et *remove* concurrentes...
- ...et consensus pour effectuer le renommage

Inadaptée aux systèmes P2P sujets au churn

[7]. ZAWIRSKI et al., « Asynchronous rebalancing of a replicated tree ».

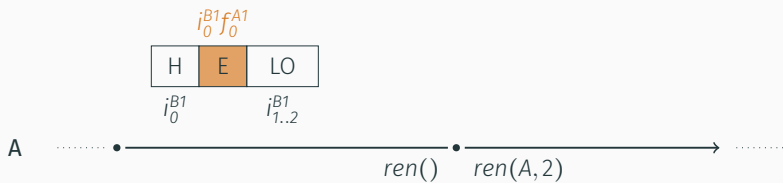
Pouvons-nous proposer un mécanisme de réduction du surcoût des CRDTs pour le type Séquence adapté aux applications LFS, c.-à-d. sans coordination synchrone ?

- CRDT pour le type Séquence qui incorpore un mécanisme de renommage
- Prend la forme d'une nouvelle opération : *rename*

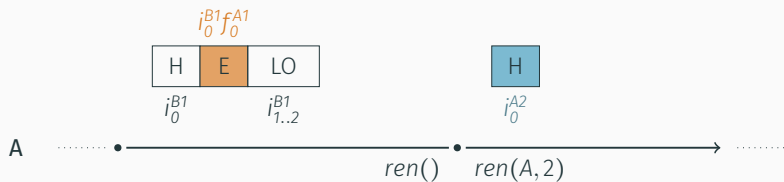
Propriétés de l'opération *rename*

- Est déterministe
- Préserve l'intention des utilisateur-rices
- Préserve la séquence, c.-à-d. unicité et ordre de ses identifiants
- Commute avec les opérations *insert*, *remove* mais aussi *rename* concurrentes

Opération *rename*

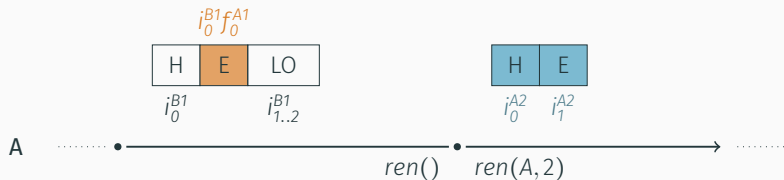


Opération *rename*



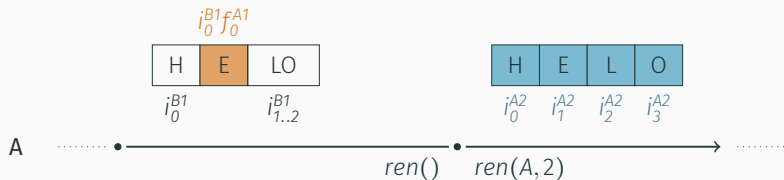
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$

Opération *rename*



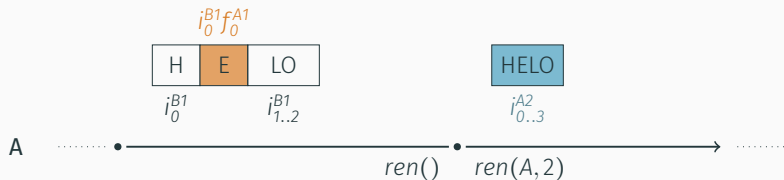
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2} , $i_2^{A2} \dots$

Opération *rename*



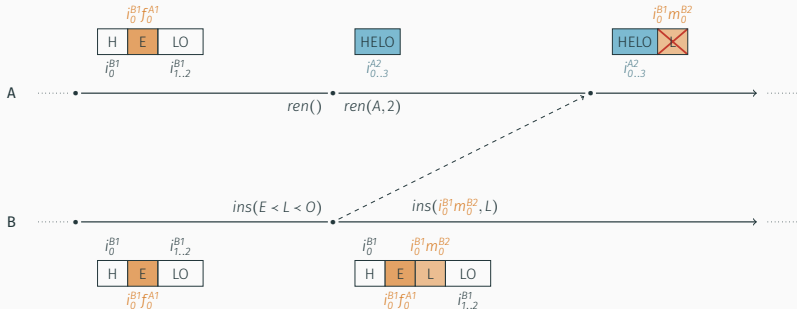
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2} , i_2^{A2} ...

Opération *rename*



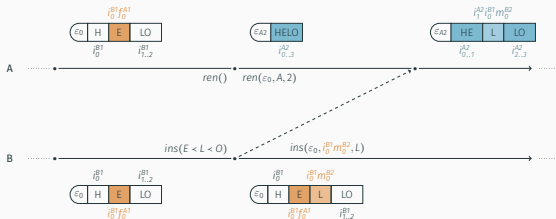
- Génère nouvel identifiant pour le 1er élément : $i_0^{B1} \rightarrow i_0^{A2}$
- Puis génère identifiants contigus pour éléments suivants : i_1^{A2} , $i_2^{A2} \dots$

Intégration des opérations *insert* et *remove* concurrentes



- Peuvent générer opérations concurrentes aux opérations *rename*
- Produisent anomalies si intégrées telles qu'elles

Correction de l'intégration des opérations concurrentes



- Ajout d'un système d'époque pour identifier les opérations concurrentes à une opération *rename*
- Transformation avant intégration des opérations concurrentes

Exemple avec $i_0^{B1} m_0^{B2}$

- Trouver son prédécesseur à l'époque d'origine ϵ_0 : $i_0^{B1} f_0^{A1}$
- Trouver son équivalent à l'époque destination ϵ_{A1} : i_1^{A2}
- Concaténer ce dernier à l'identifiant pour obtenir son équivalent à ϵ_{A1} : $i_1^{A2} i_0^{B1} m_0^{B2}$

Thanks for your attention, any questions?

