

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

THÈSE

présentée et soutenue publiquement le TODO : Définir une date

pour l'obtention du

Doctorat de l'Université de Lorraine
(mention informatique)

par

Matthieu Nicolas

Composition du jury

<i>Président :</i>	Stephan Merz
<i>Rapporteurs :</i>	Le rapporteur 1 de Paris
	Le rapporteur 2
	suite taratata
	Le rapporteur 3
<i>Examineurs :</i>	L'examineur 1 d'ici
	L'examineur 2
<i>Membres de la famille :</i>	Mon frère
	Ma sœur

Mis en page avec la classe thesul.

Remerciements

WIP

WIP

Sommaire

Introduction	1
1 Contexte	1
2 Questions de recherche et contributions	2
2.1 Ré-identification sans coordination pour Conflict-free Replicated Data Types (CRDTs) pour Séquence	2
2.2 Éditeur de texte collaboratif pair-à-pair	3
3 Plan du manuscrit	3
4 Publications	3
Chapitre 1	
Conclusions et perspectives	5
1.1 Résumé des contributions	5
1.1.1 Ré-identification sans coordination pour les CRDTs pour Séquence	5
1.1.2 Éditeur de texte collaboratif Pair-à-Pair (P2P) chiffré de bout en bout	5
1.2 Perspectives	6
1.2.1 Définition de relations de priorité pour minimiser les traitements . .	6
1.2.2 Détection et fusion manuelle de versions distantes	7
1.2.3 Étude comparative des différents modèles de synchronisation pour CRDTs	7
1.2.4 Approfondissement du patron de conception de Pure Op-based CRDTs	7
Annexe A	
Entrelacement d’insertions concurrentes dans Treedoc	
Annexe B	
Algorithmes RENAMEID	

Annexe C

Algorithmes REVERTRENAMEID

Index	15
--------------	-----------

Bibliographie

Table des figures

A.1	Modifications concurrentes d'une séquence Treedoc résultant en un entrelacement	9
-----	---	---

Introduction

1 Contexte

- Systèmes collaboratifs (wikis, plateformes de contenu, réseaux sociaux) et leurs bienfaits (qualité de l'info, vitesse de l'info (exemple de crise ?), diffusion de la parole). Démocratisation (sic) de ces systèmes au cours de la dernière décennie.
- En raison du volume de données et de requêtes, adoptent architecture décentralisée. Permet ainsi de garantir disponibilité, tolérance aux pannes et capacité de passage à l'échelle.
- Mais échoue à adresser problèmes non-techniques : confidentialité, souveraineté, protection contre censure, dépendance et nécessité de confiance envers autorité centrale.
- À l'heure où les entreprises derrière ces systèmes font preuve d'ingérence et d'intérêts contraires à ceux de leurs utilisateur·rices (Cambridge Analytica, Prism, non-modération/mise en avant de contenus racistes^{1 2 3}, invisibilisation de contenus féministes, dissolution du comité d'éthique de Google⁴, inégalité d'accès à la métamachine affectante^{5 6 7}), paraît fondamental de proposer les moyens technologiques accessibles pour concevoir et déployer des alternatives.
- *Matthieu: TODO : Voir si angle écologique/réduction consommation d'énergie peut être pertinent.*
- Systèmes pair-à-pair sont une direction intéressante pour répondre à ces problématiques, de part leur absence d'autorité centrale, la distribution des tâches et leur conception mettant le pair au centre. Mais posent de nouvelles problématiques de recherche.
- Ces systèmes ne disposent d'aucun contrôle sur les noeuds qui les composent. Le nombre de noeuds peut donc croître de manière non-bornée et atteindre des centaines de milliers de noeuds. La complexité des algorithmes de ces systèmes ne doit donc pas dépendre de ce paramètre, ou alors de manière logarithmique.

1. *Algorithms of Oppression*, Safiya Umoja Noble
2. https://www.researchgate.net/publication/342113147_The_YouTube_Algorithm_and_the_Alt-Right_Filter_Bubble
3. <https://www.wsj.com/articles/the-facebook-files-11631713039>
4. <https://www.bbc.com/news/technology-56135817>
5. *Je suis une fille sans histoire*, Alice Zeniter, p. 75
6. Qui cite *Les affects de la politique*, Frédéric Lordon
7. <https://www.bbc.com/news/technology-59011271>

- De plus, ces noeuds n'offrent aucune garantie sur leur stabilité. Ils peuvent donc rejoindre et participer au système de manière éphémère. S'agit du phénomène connu sous le nom de churn. Les algorithmes de ces systèmes ne peuvent donc pas reposer sur des mécanismes nécessitant une coordination synchrone d'une proportion des noeuds.
- Finalement, ces noeuds n'offrent aucune garanties sur leur fiabilité et intentions. Les noeuds peuvent se comporter de manière byzantine. Pour assurer la confidentialité, l'absence de confiance requise et le bon fonctionnement du système, ce dernier doit être conçu pour résister aux comportements byzantins de ses acteurs.
- Ainsi, il est nécessaire de faire progresser les technologies existantes pour les rendre compatible avec ce nouveau modèle de système. Dans le cadre de cette thèse, nous nous intéressons aux mécanismes de réplication de données dans les systèmes collaboratifs pair-à-pair temps réel.

2 Questions de recherche et contributions

2.1 Ré-identification sans coordination pour CRDTs pour Séquence

- Systèmes collaboratifs permettent aux utilisateur-rices de manipuler et éditer un contenu partagé. Pour des raisons de performance, ces systèmes autorisent généralement les utilisateur-rices à effectuer des modifications sans coordination. Leur copies divergent alors momentanément. Un mécanisme de synchronisation leur permet ensuite de récupérer l'ensemble des modifications et de les intégrer, de façon à converger. Cependant, des modifications peuvent être incompatibles entre elles, car de sémantiques contraires. Un mécanisme de résolution de conflits est alors nécessaire.
- Les CRDTs sont des types de données répliquées. Ils sont conçus pour être répliqués par les noeuds d'un système et pour permettre à ces derniers de modifier les données partagées sans aucune coordination. Dans ce but, ils incluent des mécanismes de résolution de conflits directement au sein leur spécification. Ces mécanismes leur permettent de résoudre le problème évoqué précédemment. Cependant, ces mécanismes induisent un surcoût, aussi bien d'un point de vue consommation mémoire et réseau que computationnel. Notamment, certains CRDTs comme ceux pour la Séquence souffrent d'une croissance monotone de leur surcoût. Ce surcoût s'avère handicapant dans le contexte des collaborations à large échelle.
- Pouvons-nous proposer un mécanisme sans coordination de réduction du surcoût des CRDTs pour Séquence, c.-à-d. compatible avec les systèmes pair-à-pair ?
- Dans le cadre des CRDTs pour Séquence, le surcoût du type de données répliquées provient de la croissance de leurs métadonnées. Métadonnées proviennent des identifiants associés aux éléments de la Séquence par les CRDTs. Ces identifiants sont nécessaires pour le bon fonctionnement de leur mécanisme de résolution de conflits.

- Plusieurs approches ont été proposées pour réduire le coût de ces identifiants. Notamment, [1, 2] proposent un mécanisme de ré-assignation d'identifiants de façon à réduire leur taille. Mécanisme non commutatif avec les modifications concurrentes de la Séquence, c.-à-d. l'insertion ou la suppression. Propose ainsi un mécanisme de transformation des modifications concurrentes pour gérer ces conflits. Mais mécanisme de ré-assignation n'est pas non plus commutatif avec lui-même. De fait, utilisent un algorithme de consensus pour empêcher l'exécution du mécanisme en concurrence.
- Proposons RenamableLogootSplit, un nouveau CRDT pour Séquence. Intègre un mécanisme de renommage directement au sein de sa spécification. Intègre un mécanisme de résolution de conflits pour les renommages concurrents. Permet ainsi l'utilisation du mécanisme de renommage par les noeuds sans coordination.

2.2 Éditeur de texte collaboratif pair-à-pair

- Systèmes collaboratifs adoptent généralement architecture décentralisée. Disposent d'autorités centrales qui facilitent la collaboration, l'authentification des utilisateur-ices, la communication et le stockage des données.
- Mais ces systèmes introduisent une dépendance des utilisateur-ices envers ces mêmes autorités centrales, une perte de confidentialité et de souveraineté.
- Pouvons-nous concevoir un éditeur de texte collaboratif sans autorités centrales, c.-à-d. un éditeur de texte collaboratif à large échelle pair-à-pair ?
- Ce changement de modèle, d'une architecture décentralisée à une architecture pair-à-pair, introduit un ensemble de problématiques de domaines variés, e.g.
 - (i) Comment permettre aux utilisateur-ices de collaborer en l'absence d'autorités centrales pour résoudre les conflits de modifications ?
 - (ii) Comment authentifier les utilisateur-ices en l'absence d'autorités centrales ?
 - (iii) Comment structurer le réseau de manière efficace, c.-à-d. en limitant le nombre de connexion par pair ?
- Présentons Multi User Text Editor (MUTE) [3]. S'agit, à notre connaissance, du seul prototype complet d'éditeur de texte collaboratif temps réel pair-à-pair chiffré de bout en bout. Allie ainsi les résultats issus des travaux de l'équipe sur les CRDTs pour Séquence [4, 5] et l'authentification des pairs dans systèmes distribués [6, 7] aux résultats de la littérature sur mécanismes de conscience de groupe *Matthieu: TODO : Trouver et ajouter références*, les protocoles d'appartenance aux groupe [8, 9], les réseaux pair-à-pair [10] et les protocoles d'établissement de clés de groupe [11].

3 Plan du manuscrit

4 Publications

Chapitre 1

Conclusions et perspectives

Sommaire

1.1	Résumé des contributions	5
1.1.1	Ré-identification sans coordination pour les CRDTs pour Séquence	5
1.1.2	Éditeur de texte collaboratif P2P chiffré de bout en bout	5
1.2	Perspectives	6
1.2.1	Définition de relations de priorité pour minimiser les traitements	6
1.2.2	Détection et fusion manuelle de versions distantes	7
1.2.3	Étude comparative des différents modèles de synchronisation pour CRDTs	7
1.2.4	Approfondissement du patron de conception de Pure Op-based CRDTs	7

1.1 Résumé des contributions

1.1.1 Ré-identification sans coordination pour les CRDTs pour Séquence

—

1.1.2 Éditeur de texte collaboratif P2P chiffré de bout en bout

—

1.2 Perspectives

1.2.1 Définition de relations de priorité pour minimiser les traitements

- Dans ??, nous avons spécifié la relation *priority* (cf. ??, page ??). Pour rappel, cette relation doit établir un ordre strict total sur les époques de `RenamableLogootSplit`.
- Cette relation nous permet ainsi de comparer les époques introduites par des modifications *ren* concurrentes. En l'utilisant, les noeuds peuvent ainsi déterminer vers quelle époque de l'ensemble des époques connues progresser. Cette relation permet ainsi aux noeuds de converger à une époque commune à terme.
- La convergence à terme à une époque commune présente plusieurs avantages :
 - (i) Réduire la distance entre les époques courantes des noeuds, et ainsi minimiser le surcoût en calculs par opération du mécanisme de renommage. En effet, il n'est pas nécessaire de transformer une opérations livrée avant de l'intégrer si celle-ci provient de la même époque que le noeud courant.
 - (ii) Définir un nouveau Plus Petit Ancêtre Commun (PPAC), permettant d'appliquer le mécanisme de Garbage Collection (GC) des anciens états pour minimiser le surcoût en métadonnées du mécanisme de renommage.
- Il existe plusieurs manières pour définir la relation *priority* tout en satisfaisant les propriétés indiquées. Dans le cadre de ce manuscrit, nous avons utilisé l'ordre lexicographique sur les chemins des époques dans l'*arbre des époques* pour définir *priority*. Cette approche se démarque par :
 - (i) Sa simplicité.
 - (ii) Son surcoût limité, c.-à-d. pas de métadonnées supplémentaires à stocker et diffuser, algorithme de comparaison simple.
 - (iii) Sa propriété arrangeante sur les déplacements des noeuds dans l'arbre des époques. De manière plus précise, cette définition de *priority* impose aux noeuds de se déplacer que vers l'enfant le plus à droite de l'arbre des époques. Ceci empêche les noeuds de faire un aller-retour entre deux époques données. Cette propriété permet de passer outre une contrainte concernant le couple de fonctions `renameId` et `revertRenameId` : leur réciprocity.
- Cette définition présente cependant plusieurs limites. La limite que nous identifions est sa décorrélation avec le rapport coût vs. bénéfice de progresser vers l'époque cible désignée. En effet, l'époque cible est désignée de manière arbitraire par rapport à sa position dans l'arbre des époques. Il est possible que progresser vers cette époque détériore l'état de la séquence, c.-à-d. augmente la taille des identifiants et augmente le nombre de blocs. En outre, c'est sans compter le coût en calculs induits par la transition de l'ensemble des noeuds de leur époque courante respective à cette nouvelle époque cible.
- Exemple

- Pour pallier ce problème, il est nécessaire de proposer une définition de *priority* prenant l'aspect efficacité en compte. L'approche considérée consisterait à inclure dans les opérations *ren* une ou plusieurs métriques qui représente le travail accumulé sur la branche courante de l'arbre des époques, e.g. le nombre d'opérations intégrées, les noeuds actuellement sur cette branche... L'ordre strict total entre les époques serait ainsi construit à partir de la comparaison entre les valeurs de ces métriques de leur opération *ren* respective.
- Il conviendra d'ajouter à cette nouvelle définition de *priority* un nouveau couple de fonctions `renameId` et `revertRenameId` respectant la contrainte de réciprocité de ces fonctions. Ou de mettre en place une autre implémentation du mécanisme de renommage ne nécessitant pas cette contrainte, tel que l'implémentation basée sur le journal des opérations (cf. ??, page ??).
- Il conviendra aussi d'étudier la possibilité de combiner l'utilisation de plusieurs relations *priority* pour minimiser le surcoût global du mécanisme de renommage, e.g. en fonction de la distance entre deux époques.
- Il sera nécessaire de valider l'approche proposée par une évaluation comparative par rapport à l'approche actuelle. Elle consistera à monitorer le coût du système pour observer si l'approche proposée permet de réduire les calculs de manière globale. Plusieurs configurations de paramètres pourront aussi être utilisées pour déterminer l'impact respectif de chaque paramètre sur les résultats.

1.2.2 Détection et fusion manuelle de versions distantes

- Est-ce que ça a vraiment du sens d'intégrer automatiquement des modifications ayant été générées sur une version du document distante de l'état actuel du document (voir distance de Hamming, Levenshtein, String-to-string correction problem (Tichy et al))
- Jusqu'à quelle distance est-ce que la fusion automatique a encore du sens ? *Matthieu: NOTE : Peut connecter ça à la nécessité de conserver un chemin d'une époque à l'autre : si les opérations émises depuis cette époque ont probablement plus d'intérêt pour l'état actuel, couper l'arbre ?*

1.2.3 Étude comparative des différents modèles de synchronisation pour CRDTs

1.2.4 Approfondissement du patron de conception de Pure Op-based CRDTs

Annexe A

Entrelacement d'insertions concurrentes dans Treedoc

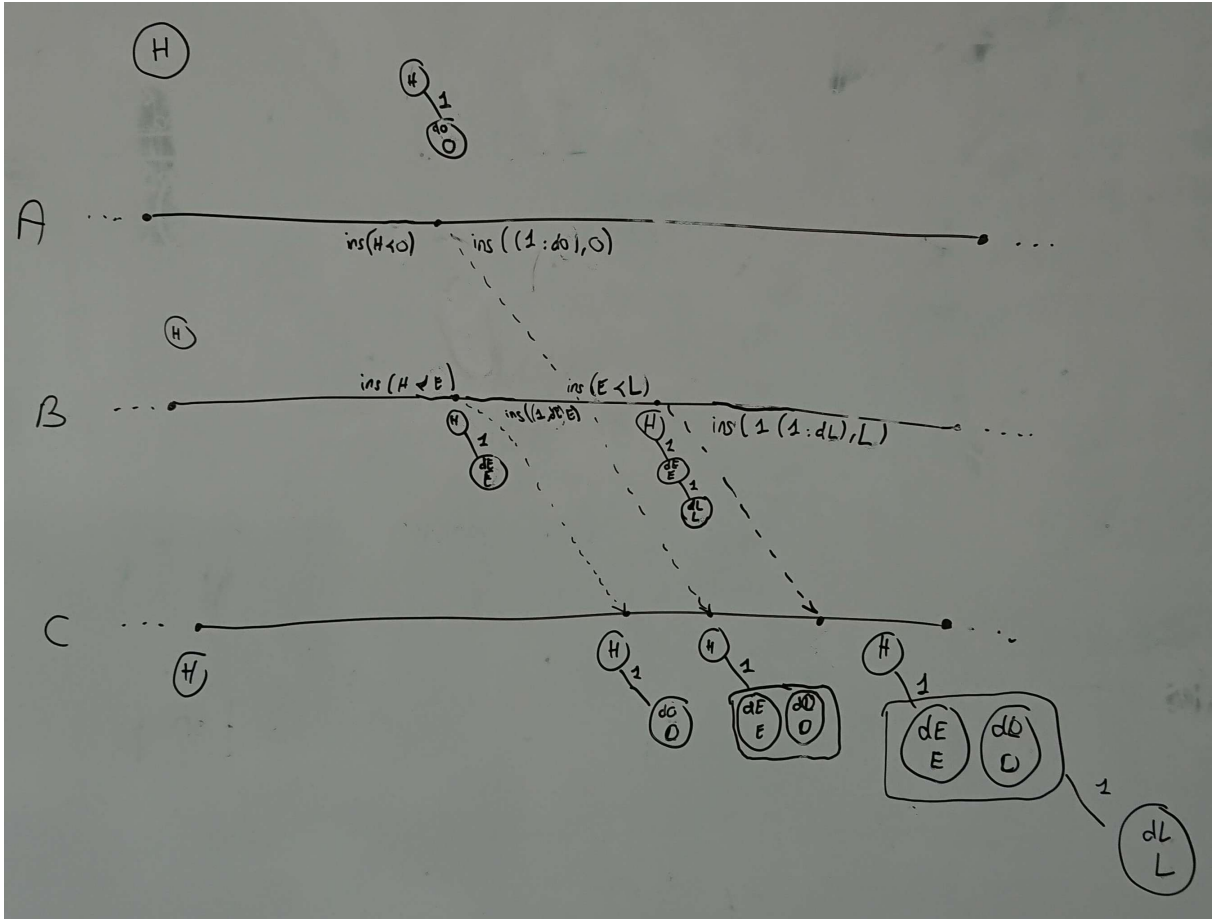


FIGURE A.1 – Modifications concurrentes d'une séquence Treedoc résultant en un entrelacement

Matthieu: TODO : Réaliser au propre contre-exemple. Nécessite que $d_E < d_O$, inverser A et B histoire d'éviter toute confusion. En soi, C pas nécessaire, à voir si le conserve.

Annexe B

Algorithmes RENAMEID

Algorithme 1 Remaining functions to rename an identifier

```
function RENIDLESTHANFIRSTID(id, newFirstId)
  if id < newFirstId then
    return id
  else
    pos ← position(newFirstId)
    nId ← nodeId(newFirstId)
    nSeq ← nodeSeq(newFirstId)
    predNewFirstId ← new Id(pos, nId, nSeq, -1)

    return concat(predNewFirstId, id)
  end if
end function

function RENIDGREATERTHANLASTID(id, newLastId)
  if id < newLastId then
    return concat(newLastId, id)
  else
    return id
  end if
end function
```

Annexe C

Algorithmes REVERTRENAMEID

Algorithme 2 Remaining functions to revert an identifier renaming

```
function REVRENIDLESTHANNEWFIRSTID(id, firstId, newFirstId)
  predNewFirstId  $\leftarrow$  createIdFromBase(newFirstId, -1)
  if isPrefix(predNewFirstId, id) then
    tail  $\leftarrow$  getTail(id, 1)
    if tail < firstId then
      return tail
    else
      ▷ id has been inserted causally after the rename op
      offset  $\leftarrow$  getLastOffset(firstId)
      predFirstId  $\leftarrow$  createIdFromBase(firstId, offset)
      return concat(predFirstId, MAX_TUPLE, tail)
    end if
  else
    return id
  end if
end function

function REVRENIDGREATERTHANNEWLASTID(id, lastId)
  if id < lastId then
    ▷ id has been inserted causally after the rename op
    return concat(lastId, MIN_TUPLE, id)
  else if isPrefix(newLastId, id) then
    tail  $\leftarrow$  getTail(id, 1)
    if tail < lastId then
      ▷ id has been inserted causally after the rename op
      return concat(lastId, MIN_TUPLE, tail)
    else if tail < newLastId then
      return tail
    else
      ▷ id has been inserted causally after the rename op
      return id
    end if
  else
    return id
  end if
end function
```

Index

Voici un index

FiXme :

Notes :

- 1 : Matthieu : TODO : Voir si angle écologique/réduction consommation d'énergie peut être pertinent., 1
- 2 : Matthieu : TODO : Trouver et ajouter références, 3
- 3 : Matthieu : NOTE : Peut connecter ça à la nécessité de conserver un chemin d'une époque à l'autre : si les opérations émises depuis cette époque ont probablement plus d'intérêt pour l'état actuel, couper l'arbre ?, 7
- 4 : Matthieu : TODO : Réaliser au propre contre-exemple. Nécessite que $d_E < d_O$, inverser A et B histoire d'éviter toute confusion. En soi, C pas nécessaire, à voir si le conserve. , 9

FiXme (Matthieu) :

Notes :

- 1 : TODO : Voir si angle écologique/réduction consommation d'énergie peut être pertinent., 1
- 2 : TODO : Trouver et ajouter références, 3
- 3 : NOTE : Peut connecter ça à la nécessité de conserver un chemin d'une époque à l'autre : si les opérations émises depuis cette époque ont probablement plus d'intérêt pour l'état actuel, couper l'arbre ?, 7
- 4 : TODO : Réaliser au propre contre-exemple. Nécessite que $d_E < d_O$, in-

verser A et B histoire d'éviter toute confusion. En soi, C pas nécessaire, à voir si le conserve. , 9

Bibliographie

- [1] Mihai LETIA, Nuno PREGUIÇA et Marc SHAPIRO. « Consistency without concurrency control in large, dynamic systems ». In : *LADIS 2009 - 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware*. T. 44. Operating Systems Review 2. Big Sky, MT, United States : Assoc. for Computing Machinery, oct. 2009, p. 29–34. DOI : 10.1145/1773912.1773921. URL : <https://hal.inria.fr/hal-01248270>.
- [2] Marek ZAWIRSKI, Marc SHAPIRO et Nuno PREGUIÇA. « Asynchronous rebalancing of a replicated tree ». In : *Conférence Française en Systèmes d'Exploitation (CFSE)*. Saint-Malo, France, mai 2011, p. 12. URL : <https://hal.inria.fr/hal-01248197>.
- [3] Matthieu NICOLAS, Victorien ELVINGER, Gérald OSTER, Claudia-Lavinia IGNAT et François CHAROY. « MUTE : A Peer-to-Peer Web-based Real-time Collaborative Editor ». In : *ECSCW 2017 - 15th European Conference on Computer-Supported Cooperative Work*. T. 1. Proceedings of 15th European Conference on Computer-Supported Cooperative Work - Panels, Posters and Demos 3. Sheffield, United Kingdom : EUSSET, août 2017, p. 1–4. DOI : 10.18420/ecscw2017_p5. URL : <https://hal.inria.fr/hal-01655438>.
- [4] Luc ANDRÉ, Stéphane MARTIN, Gérald OSTER et Claudia-Lavinia IGNAT. « Supporting Adaptable Granularity of Changes for Massive-Scale Collaborative Editing ». In : *International Conference on Collaborative Computing : Networking, Applications and Worksharing - CollaborateCom 2013*. Austin, TX, USA : IEEE Computer Society, oct. 2013, p. 50–59. DOI : 10.4108/icst.collaboratecom.2013.254123.
- [5] Victorien ELVINGER. « Réplication sécurisée dans les infrastructures pair-à-pair de collaboration ». Theses. Université de Lorraine, juin 2021. URL : <https://hal.univ-lorraine.fr/tel-03284806>.
- [6] Hoang-Long NGUYEN, Claudia-Lavinia IGNAT et Olivier PERRIN. « Trusternity : Auditing Transparent Log Server with Blockchain ». In : *Companion of the The Web Conference 2018*. Lyon, France, avr. 2018. DOI : 10.1145/3184558.3186938. URL : <https://hal.inria.fr/hal-01883589>.
- [7] Hoang-Long NGUYEN, Jean-Philippe EISENBARTH, Claudia-Lavinia IGNAT et Olivier PERRIN. « Blockchain-Based Auditing of Transparent Log Servers ». In : *32th IFIP Annual Conference on Data and Applications Security and Privacy (DBSec)*. Sous la dir. de Florian KERSCHBAUM et Stefano PARABOSCHI. T. LNCS-10980.

- Data and Applications Security and Privacy XXXII. Part 1 : Administration. Bergamo, Italy : Springer International Publishing, juil. 2018, p. 21–37. DOI : 10.1007/978-3-319-95729-6_2. URL : <https://hal.archives-ouvertes.fr/hal-01917636>.
- [8] Abhinandan DAS, Indranil GUPTA et Ashish MOTIVALA. « SWIM : scalable weakly-consistent infection-style process group membership protocol ». In : *Proceedings International Conference on Dependable Systems and Networks*. 2002, p. 303–312. DOI : 10.1109/DSN.2002.1028914.
- [9] Armon DADGAR, James PHILLIPS et Jon CURREY. « Lifeguard : Local health awareness for more accurate failure detection ». In : *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2018, p. 22–25.
- [10] Brice NÉDELEC, Julian TANKE, Davide FREY, Pascal MOLLI et Achour MOSTÉFAOUI. « An adaptive peer-sampling protocol for building networks of browsers ». In : *World Wide Web* 21.3 (2018), p. 629–661.
- [11] Mike BURMESTER et Yvo DESMEDT. « A secure and efficient conference key distribution system ». In : *Advances in Cryptology — EUROCRYPT’94*. Sous la dir. d’Alfredo DE SANTIS. Berlin, Heidelberg : Springer Berlin Heidelberg, 1995, p. 275–286. ISBN : 978-3-540-44717-7.

Résumé

Afin d'assurer leur haute disponibilité, les systèmes distribués à large échelle se doivent de répliquer leurs données tout en minimisant les coordinations nécessaires entre noeuds. Pour concevoir de tels systèmes, la littérature et l'industrie adoptent de plus en plus l'utilisation de types de données répliquées sans conflits (CRDTs). Les CRDTs sont des types de données qui offrent des comportements similaires aux types existants, tel l'Ensemble ou la Séquence. Ils se distinguent cependant des types traditionnels par leur spécification, qui supporte nativement les modifications concurrentes. À cette fin, les CRDTs incorporent un mécanisme de résolution de conflits au sein de leur spécification.

Afin de résoudre les conflits de manière déterministe, les CRDTs associent généralement des identifiants aux éléments stockés au sein de la structure de données. Les identifiants doivent respecter un ensemble de contraintes en fonction du CRDT, telles que l'unicité ou l'appartenance à un ordre dense. Ces contraintes empêchent de borner la taille des identifiants. La taille des identifiants utilisés croît alors continuellement avec le nombre de modifications effectuées, aggravant le surcoût lié à l'utilisation des CRDTs par rapport aux structures de données traditionnelles. Le but de cette thèse est de proposer des solutions pour pallier ce problème.

Nous présentons dans cette thèse deux contributions visant à répondre à ce problème : (i) Un nouveau CRDT pour Séquence, *RenamableLogootSplit*, qui intègre un mécanisme de renommage à sa spécification. Ce mécanisme de renommage permet aux noeuds du système de réattribuer des identifiants de taille minimale aux éléments de la séquence. Cependant, cette première version requiert une coordination entre les noeuds pour effectuer un renommage. L'évaluation expérimentale montre que le mécanisme de renommage permet de réinitialiser à chaque renommage le surcoût lié à l'utilisation du CRDT. (ii) Une seconde version de *RenamableLogootSplit* conçue pour une utilisation dans un système distribué. Cette nouvelle version permet aux noeuds de déclencher un renommage sans coordination préalable. L'évaluation expérimentale montre que cette nouvelle version présente un surcoût temporaire en cas de renommages concurrents, mais que ce surcoût est à terme.

Mots-clés: CRDTs, édition collaborative en temps réel, cohérence à terme, optimisation mémoire, performance

Abstract

Keywords: CRDTs, real-time collaborative editing, eventual consistency, memory-wise optimisation, performance

