UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Ré-identification efficace dans les types de données répliquées sans conflit (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

Président :	Stephan Merz	
Rapporteurs :	Le rapporteur 1	de Paris
	Le rapporteur 2	
	suite	taratata
	Le rapporteur 3	
Examinateurs :	L'examinateur 1	d'ici
	L'examinateur 2	
Membres de la famille :	Mon frère	
	Ma sœur	

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Les remerciements.

Je dédie cette thèse
à ma machine.
Oui, à Pandore,
qui fut la première de toutes.

Sommaire

Table des figures

Introduction

Chapitre 1

État de l'art

Sommaire

1.1 Transformées opérationnelles

1.2 Séquences répliquées sans conflits

1.2.1 Types de données répliquées sans conflits

Principes

— Nouvelles spécifications des types de données existants

— Structures conçues pour être répliquées au sein d'un système

— Et être modifiées sans coordination par ses différents noeuds

— Doivent donc supporter de nouveaux scénarios uniquement possible dans des exécutions parallèles

— Et définir une sémantique pour ces scénarios inédits

  - Exemple du Registre avec LWW-Register et MV-Register ?
- Pour gérer ces scénarios, intègrent un mécanisme de résolution de conflits directement au sein de leur spécification
- Garantissent la cohérence forte à terme

Familles de types de données répliquées sans conflits

- Une catégorisation des CRDTs a été proposée
- Propose de répartir les CRDTs en différentes familles en fonction de la méthode de synchronisation utilisée
- Chacune de ces méthodes de synchronisation implique des contraintes sur la couche réseau du système et entraîne des répercussions sur la structure de données elle-même
- Types de données répliquées sans conflits à base d'états [28, 27]
  - Les noeuds partagent leur état de manière périodique
  - Une fonction *merge* permet aux noeuds de fusionner leur état courant avec un autre état reçu
  - Aucune hypothèse sur la partie réseau autre que les noeuds arrivent à communiquer à terme
  - Pas un problème si états perdus, les prochains intégreront les informations de ces derniers
  - Pas un problème si états reçus dans le désordre, la fonction *merge* est commutative
  - Pas un problème si états reçus plusieurs fois, *merge* est idempotent
  - Mais nécessite de conserver au sein de la structure de données assez d'informations pour proposer une telle fonction de *merge*
  - Par exemple, besoin de conserver une trace des éléments supprimés pour empêcher leur réapparition suite à une fusion d'états
  - *Matthieu: TODO : Ajouter forces, faiblesses et cas d'utilisation de cette approche*
- Types de données répliquées sans conflits à base d'opérations [28, 27, 6, 5]
  - Les noeuds partagent uniquement des opérations représentant leurs modifications
  - Une modification peut se formaliser en deux étapes
  - *prepare*, qui permet de générer une opération correspondant à une modification
  - *effect*, qui permet d'appliquer l'effet de la modification à un état
  - Les opérations concurrentes doivent être commutatives pour assurer la convergence
  - Mais pas de contraintes sur les opérations causalement liées

  - Pas de contraintes non plus sur l'idempotence des opérations
  - Nécessite donc généralement d'ajouter une couche *livraison* pour faire le lien entre le réseau et le CRDT
  - Permet d'attacher des informations de causalité aux opérations locales avant de les envoyer
  - Permet de ré-ordonner et filtrer les opérations distantes reçues avant de les fournir au CRDT
  - Besoin d'un mécanisme d'anti-entropie [23] pour assurer que l'ensemble des noeuds observent l'ensemble des opérations et ainsi garantir la convergence
  - Permet de lisser la consommation réseau
  - Offre des temps d'intégration et de propagation des modifications rapides
  - Mais accumule des métadonnées puisque les noeuds doivent conserver les opérations passées pour permettre à un nouveau noeud de rejoindre la collaboration et de se synchroniser
  - Possible de tronquer le log des opérations en se basant sur la stabilité causale [7] afin de limiter cette accumulation de métadonnées
- Types de données répliquées sans conflits à base de différences [3, 2]

Adoption dans la littérature et l'industrie

- Conception et développement de librairies mettant à disposition des développeurs d'applications des types de données composés [18, 17, 33, 13, 4]
- Conception de langages de programmation intégrant des CRDTs comme types primitifs, destinés au développement d'applications distribuées [15, 11]
- Conception et implémentation de bases de données distribuées, relationnelles ou non, privilégiant la disponibilité et la minimisation de la latence à l'aide des CRDTs [25, 10, 32, 9, 34]
- Conception d'un nouveau paradigme d'applications, Local-First Software, dont une des fondations est les CRDTs [14, 12]
- Éditeurs collaboratifs temps réel à large échelle et offrant de nouveaux scénarios de collaboration grâce aux CRDTs [16, 21]

1.2.2 Approches pour les séquences répliquées sans conflits

Approche à pierres tombales

- WOOT [22, 31, 1]
- RGA [26]
- RGASplit [8]

Approche à identifiants densément ordonnés

— Treedoc [24]
— Logoot [29, 30]

1.3 LogootSplit

1.3.1 Identifiants

Composition

Stratégie d'allocation

1.3.2 Aggrégation dynamique d'élements en blocs

1.3.3 Limites

1.4 Mitigation du surcoût des séquences répliquées sans conflits

1.4.1 Core-Nebula

1.4.2 LSEQ

Matthieu: Serait intéressant d'avoir une implémentation combinant LogootSplit et LSEQ pour vérifier si les contraintes sur la création de blocs dans LogootSplit ne "sabotent" pas la croissance polylogarithmique des identifiants de LSEQ

1.5 Synthèse

Chapitre 2

Présentation de l'approche

Sommaire

Nous proposons un nouveau Conflict-free Replicated Data Type (CRDT) pour la *Sequence* appartenant à l'approche des identifiants densément ordonnées : RenamableLogootSplit [19, 20]. Cette structure de données permet aux pairs d'insérer et de supprimer des éléments au sein d'une séquence répliquée. Nous introduisons une opération de renommage qui permet de 1. réassigner des identifiants plus courts aux différents éléments de la séquence 2. fusionner les blocs composant la séquence. Ces deux actions permettent à l'opération de renommage de produire un nouvel état minimisant son surcoût en métadonnées.

2.1 Modèle du système

Le système est composé d'un ensemble dynamique de noeuds, les noeuds pouvant rejoindre puis quitter la collaboration tout au long de sa durée. Les noeuds collaborent afin de construire et maintenir une séquence à l'aide de RenamableLogootSplit. Chaque noeud possède une copie de la séquence et peut l'éditer sans se coordonner avec les autres. Les modifications des noeuds prennent la forme d'opérations qui sont appliquées immédiatement à leur copie locale. Les opérations sont ensuite transmises de manière asynchrone aux autres noeudds pour qu'ils puissent à leur tour appliquer les modifications à leur copie.

Les noeuds communiquement par l'intermédiaire d'un réseau Pair-à-Pair (P2P). Ce réseau est non-fiable : les messages peuvent être perdus, ré-ordonnés ou même livrés à plusieurs reprises. Le réseau peut aussi être sujet à des partitions, qui séparent alors les noeuds en des sous-groupes disjoints. Afin de compenser les limitations du réseau, les noeuds reposent sur une couche de livraison de messages.

Puisque RenamableLogootSplit est une extension de LogootSplit, il partage les mêmes contraintes sur la livraison de messages. La couche de livraison de messages sert donc à livrer les messages à l'application exactement une fois. La couche de livraison de messages a aussi pour tâche de garantir la livraison des opérations de suppression après les opérations d'insertion correspondantes. Finalement, la couche de livraison intègre aussi un mécanisme d'anti-entropie [23]. Ce mécanisme permet aux noeuds de se synchroniser par paires, en détectant et ré-échangeant les messages perdus.

2.2 Définition de l'opération de renommage

2.2.1 Objectifs

2.2.2 Propriétés

2.2.3 Contraintes

Chapitre 3

Renommage dans un système centralisé

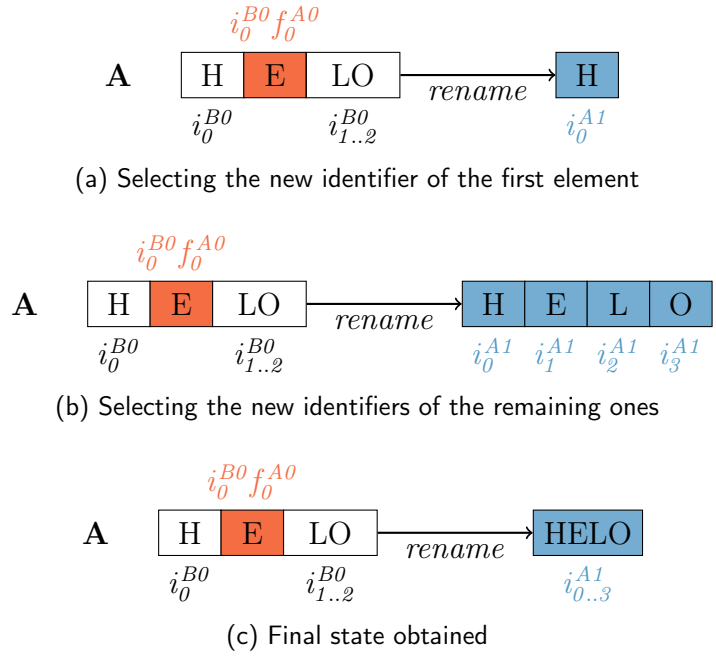
Sommaire

3.1 RenamableLogootSplit

3.1.1 Opération de renommage proposée

Notre opération de renommage permet à RenamableLogootSplit de réduire le surcoût en métadonnées des séquences répliquées. Pour ce faire, elle réassigne des identifiants arbitraires aux éléments de la séquence.

Son comportement est illustré dans la Figure 3.1. Dans cet exemple, le noeud A initie une opération *rename* sur son état local. Tout d'abord, le noeud A réutilise l'identifiant du premier élément de la séquence (i_0^{B0}) mais en le modifiant avec son propre identifiant de noeud (**A**) et numéro de séquence actuel (*1*). De plus, son offset est mis à 0. Le noeud A réassigne l'identifiant résultant (i_0^{A1}) au premier élément de la séquence, comme décrit



(a) Selecting the new identifier of the first element

(b) Selecting the new identifiers of the remaining ones

(c) Final state obtained

FIGURE 3.1 – Renaming the sequence on node A

dans 3.1a. Ensuite, le noeud A dérive des identifiants contigus pour tous les éléments restants en incrémentant de manière successive l'offset (i_1^{A1}, i_2^{A1}, i_3^{A1}), comme présenté dans 3.1b. Comme nous assignons des identifiants consécutifs à tous les éléments de la séquence, nous pouvons au final aggréger ces éléments en un seul bloc, comme illustré en 3.1c. Ceci permet aux noeuds de bénéficier au mieux de la fonctionnalité des blocs et de minimiser le surcoût en métadonnés de l'état résultat.

3.1.2 Gestion des opérations concurrentes au renommage

3.1.3 Processus d'intégration d'une opération

3.1.4 Évolution du modèle de cohérence

Matthieu: TODO : Revoir si une livraison causale de l'opération rename *(et non epoch-based) peut avoir un intérêt*

3.1.5 Récupération de la mémoire des états précédents

3.2 Validation

3.2.1 Preuve de correction

3.2.2 Complexité temporelle

3.3 Discussion

3.3.1 Stockage des états précédents sur disque

3.3.2 Utilisation de l'opération de renommage comme snapshot

- L'opération de renommage embarque la somme de toutes les opérations passées sous la forme du *former state*
- On peut à tout moment re-calculer l'état courant du document à partir de l'opération de renommage primaire, des opérations concurrentes à cette dernière et des opérations générées depuis
- Peut utiliser ce principe pour le mécanisme de synchronisation
- Lorsqu'un nouveau pair rejoint la collaboration, le noeud avec lequel il se synchronise peut lui fournir uniquement ce sous-ensemble des opérations
- De la même manière, on pourrait généraliser l'utilisation de cette méthode de synchronisation
- À la réception d'une demande de synchronisation d'un pair présentant un important retard, le noeud peut choisir d'employer cette méthode
  - Plutôt que de lui envoyer et de lui faire rejouer l'ensemble des opérations
- La question étant de comment procéder pour quantifier ce retard et pour définir le seuil à partir duquel ce retard est considéré comme important
- Cette méthode de synchronisation pose néanmoins le problème suivant
- Le pair synchronisé de cette manière ne possède qu'une partie du log des opérations
- S'il reçoit ensuite une demande de synchronisation d'un autre pair, il est possible qu'il ne puisse y répondre
  - Cas où il manque à l'autre pair juste une opération d'avant le renommage (possible si les dépendances causales ne sont pas requises pour intégrer l'opération de renommage)
- Dans ce cas, ne peut pas fournir la seule opération manquante au pair qui la demande
  - *Matthieu: NOTE : Mais dans ce cas, le pair peut tout à fait générer un état courant à jour à partir des infos qu'il possède puisque l'opération qui lui manque est intégrée dans l'opé de renommage*
- *Matthieu: TODO : Étudier si y a un intérêt à privilégier la synchronisation basée sur l'intégration successive de toutes les opérations quand on a cette méthode de synchronisation par snapshot/checkpoint de possible*

3.3.3 Compression de l'opération de renommage

3.3.4 Limitation de la taille de l'opération de renommage

3.4 Conclusion

Chapitre 4

Renommage dans un système distribué

Sommaire

4.1 RenamableLogootSplit v2

4.1.1 Conflits en cas de renommages concurrents

4.1.2 Méthode de résolution de conflits proposée

4.1.3 Relation de priorité entre renommages

4.1.4 Algorithme d'annulation de l'opération de renommage

4.1.5 Mise à jour du processus d'intégration d'une opération

4.1.6 Mise à jour des règles de récupération de la mémoire des états précédents

4.2 Validation

4.2.1 Complexité temporelle

4.2.2 Expérimentations

Scénario d'expérimentation

Implémentation des simulations

4.2.3 Résultats

Convergence

Consommation mémoire

Temps d'intégration des opérations "simples"

Temps d'intégration de l'opération de renommage

4.3 Discussion

4.3.1 Implémentation alternative à base d'operation-log

4.3.2 Définition de relations de priorité plus optimales

4.3.3 Report de la transition vers la nouvelle epoch principale

4.4 Conclusion

Chapitre 5

Stratégies de déclenchement du renommage

Sommaire

5.1 Motivation

5.2 Stratégies proposées

5.2.1 Propriétés

5.2.2 Stratégie 1 : ? ? ?

5.2.3 Stratégie 2 : ? ? ?

NOTE : Peut considérer une stratégie où on prend en compte la hauteur de l'epoch tree pour déclencher un renommage : peut attendre qu'on ait plus qu'une epoch pour autoriser un nouveau renommage d'avoir lieu. Permettrait d'empêcher le cas où un noeud revient après 6 mois d'absence et doit intégrer 100 renommages avant de pouvoir collaborer. Mais dans le cas où un noeud ne rejoint plus la collaboration, bloque le mécanisme de renommage pour les autres

5.3 Évaluation

5.4 Conclusion

Chapitre 6

Conclusions et perspectives

Sommaire

6.1 Résumé des contributions

6.2 Perspectives

6.2.1 Définition de relations de priorité plus optimales

6.2.2 Redéfinition de la sémantique du renommage en déplacement d'éléments

6.2.3 Définition de types de données répliquées sans conflits plus complexes

Annexe A

Algorithmes

Index

Voici un index

Bibliographie

[1] Mehdi Ahmed-Nacer et al. « Evaluating CRDTs for Real-time Document Editing ». In : *11th ACM Symposium on Document Engineering*. Sous la dir. d'ACM. Mountain View, California, United States, sept. 2011, p. 103–112. DOI : 10.1145/2034691.2034717. URL : https://hal.inria.fr/inria-00629503.

[2] Paulo Sérgio Almeida, Ali Shoker et Carlos Baquero. « Delta state replicated data types ». In : *Journal of Parallel and Distributed Computing* 111 (jan. 2018), p. 162–173. ISSN : 0743-7315. DOI : 10.1016/j.jpdc.2017.08.003. URL : http://dx.doi.org/10.1016/j.jpdc.2017.08.003.

[3] Paulo Sérgio Almeida, Ali Shoker et Carlos Baquero. « Efficient State-Based CRDTs by Delta-Mutation ». In : *Networked Systems*. Sous la dir. d'Ahmed Bouajjani et Hugues Fauconnier. Cham : Springer International Publishing, 2015, p. 62–76. ISBN : 978-3-319-26850-7.

[4] Automerge. *Automerge : data structures for building collaborative applications in Javascript*. URL : https://github.com/automerge/automerge.

[5] Carlos Baquero, Paulo Sergio Almeida et Ali Shoker. *Pure Operation-Based Replicated Data Types*. 2017. arXiv : 1710.04469 [cs.DC].

[6] Carlos Baquero, Paulo Sérgio Almeida et Ali Shoker. « Making Operation-Based CRDTs Operation-Based ». In : *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*. PaPEC '14. Amsterdam, The Netherlands : Association for Computing Machinery, 2014. ISBN : 9781450327169. DOI : 10.1145/2596631.2596632. URL : https://doi.org/10.1145/2596631.2596632.

[7] Carlos Baquero, Paulo Sérgio Almeida et Ali Shoker. « Making Operation-Based CRDTs Operation-Based ». In : *Distributed Applications and Interoperable Systems*. Sous la dir. de Kostas Magoutis et Peter Pietzuch. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, p. 126–140.

[8] Loïck Briot, Pascal Urso et Marc Shapiro. « High Responsiveness for Group Editing CRDTs ». In : *ACM International Conference on Supporting Group Work*. Sanibel Island, FL, United States, nov. 2016. DOI : 10.1145/2957276.2957300. URL : https://hal.inria.fr/hal-01343941.

[9] Concordant. *Concordant*. URL : http://www.concordant.io/.

[10] The SyncFree Consortium. *AntidoteDB : A planet scale, highly available, transactional database*. URL : http://antidoteDB.eu/.

[11] Kevin De Porre et al. « CScript : A distributed programming language for building mixed-consistency applications ». In : *Journal of Parallel and Distributed Computing volume 144* (oct. 2020), p. 109–123. ISSN : 0743-7315. DOI : 10.1016/j.jpdc.2020.05.010.

[12] Peter van Hardenberg et Martin Kleppmann. « PushPin : Towards Production-Quality Peer-to-Peer Collaboration ». In : *7th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC 2020. ACM, avr. 2020. DOI : 10.1145/3380787.3393683.

[13] Martin Kleppmann et Alastair R. Beresford. « A Conflict-Free Replicated JSON Datatype ». In : *IEEE Transactions on Parallel and Distributed Systems* 28.10 (oct. 2017), p. 2733–2746. ISSN : 1045-9219. DOI : 10.1109/tpds.2017.2697382. URL : http://dx.doi.org/10.1109/TPDS.2017.2697382.

[14] 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. Athens, Greece : Association for Computing Machinery, 2019, p. 154–178. ISBN : 9781450369954. DOI : 10.1145/3359591.3359737. URL : https://doi.org/10.1145/3359591.3359737.

[15] Christopher Meiklejohn et Peter Van Roy. « Lasp : A Language for Distributed, Coordination-free Programming ». In : *17th International Symposium on Principles and Practice of Declarative Programming*. PPDP 2015. ACM, juil. 2015, p. 184–195. DOI : 10.1145/2790449.2790525.

[16] Brice Nédelec, Pascal Molli et Achour Mostefaoui. « CRATE : Writing Stories Together with our Browsers ». In : *25th International World Wide Web Conference*. WWW 2016. ACM, avr. 2016, p. 231–234. DOI : 10.1145/2872518.2890539.

[17] Petru Nicolaescu et al. « Near Real-Time Peer-to-Peer Shared Editing on Extensible Data Types ». In : *19th International Conference on Supporting Group Work*. GROUP 2016. ACM, nov. 2016, p. 39–49. DOI : 10.1145/2957276.2957310.

[18] Petru Nicolaescu et al. « Yjs : A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types ». In : *15th International Conference on Web Engineering*. ICWE 2015. Springer LNCS volume 9114, juin 2015, p. 675–678. DOI : 10.1007/978-3-319-19890-3_55. URL : http://dbis.rwth-aachen.de/~derntl/papers/preprints/icwe2015-preprint.pdf.

[19] Matthieu Nicolas. « Efficient renaming in CRDTs ». In : *Middleware 2018 - 19th ACM/IFIP International Middleware Conference (Doctoral Symposium)*. Rennes, France, déc. 2018. URL : https://hal.inria.fr/hal-01932552.

[20] Matthieu Nicolas, Gérald Oster et Olivier Perrin. « Efficient Renaming in Sequence CRDTs ». In : *7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC'20)*. Heraklion, Greece, avr. 2020. URL : https://hal.inria.fr/hal-02526724.

[21] Matthieu NICOLAS et al. « 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.

[22] Gérald OSTER et al. « Data Consistency for P2P Collaborative Editing ». In : *ACM Conference on Computer-Supported Cooperative Work - CSCW 2006*. Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. Banff, Alberta, Canada : ACM Press, nov. 2006, p. 259–268. URL : https://hal.inria.fr/inria-00108523.

[23] D. S. PARKER et al. « Detection of Mutual Inconsistency in Distributed Systems ». In : *IEEE Trans. Softw. Eng.* 9.3 (mai 1983), p. 240–247. ISSN : 0098-5589. DOI : 10.1109/TSE.1983.236733. URL : https://doi.org/10.1109/TSE.1983.236733.

[24] Nuno PREGUICA et al. « A Commutative Replicated Data Type for Cooperative Editing ». In : *2009 29th IEEE International Conference on Distributed Computing Systems*. Juin 2009, p. 395–403. DOI : 10.1109/ICDCS.2009.20.

[25] RIAK. *Riak KV*. URL : http://riak.com/.

[26] Hyun-Gul ROH et al. « Replicated abstract data types : Building blocks for collaborative applications ». In : *Journal of Parallel and Distributed Computing* 71.3 (2011), p. 354–368. ISSN : 0743-7315. DOI : https://doi.org/10.1016/j.jpdc.2010.12.006. URL : http://www.sciencedirect.com/science/article/pii/S0743731510002716.

[27] Marc SHAPIRO et al. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Research Report RR-7506. Inria – Centre Paris-Rocquencourt ; INRIA, jan. 2011, p. 50. URL : https://hal.inria.fr/inria-00555588.

[28] Marc SHAPIRO et al. « Conflict-Free Replicated Data Types ». In : *Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems*. SSS 2011. 2011, p. 386–400. DOI : 10.1007/978-3-642-24550-3_29.

[29] Stéphane WEISS, Pascal URSO et Pascal MOLLI. « Logoot : A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks ». In : *Proceedings of the 29th International Conference on Distributed Computing Systems - ICDCS 2009*. Montreal, QC, Canada : IEEE Computer Society, juin 2009, p. 404–412. DOI : 10.1109/ICDCS.2009.75. URL : http://doi.ieeecomputersociety.org/10.1109/ICDCS.2009.75.

[30] Stéphane WEISS, Pascal URSO et Pascal MOLLI. « Logoot-Undo : Distributed Collaborative Editing System on P2P Networks ». In : *IEEE Transactions on Parallel and Distributed Systems* 21.8 (août 2010), p. 1162–1174. DOI : 10.1109/TPDS.2009.173. URL : https://hal.archives-ouvertes.fr/hal-00450416.

[31] Stéphane WEISS, Pascal URSO et Pascal MOLLI. « Wooki : a P2P Wiki-based Collaborative Writing Tool ». In : t. 4831. Déc. 2007. ISBN : 978-3-540-76992-7. DOI : 10.1007/978-3-540-76993-4_42.

[32] C. Wu et al. « Anna : A KVS for Any Scale ». In : *IEEE Transactions on Knowledge and Data Engineering* 33.2 (2021), p. 344–358. DOI : 10.1109/TKDE.2019.2898401.

[33] Yjs. *Yjs : A CRDT framework with a powerful abstraction of shared data.* URL : https://github.com/yjs/yjs.

[34] Weihai Yu et Claudia-Lavinia Ignat. « Conflict-Free Replicated Relations for Multi-Synchronous Database Management at Edge ». In : *IEEE International Conference on Smart Data Services, 2020 IEEE World Congress on Services*. Beijing, China, oct. 2020. URL : https://hal.inria.fr/hal-02983557.

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