UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

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

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

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 Google [4], inégalité d'accès à la méta-machine affectante [5] [6] [7]), parait 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 éphèmère. 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 mécanimes 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-rices, la communication et le stockage des données.

— Mais ces systèmes introduisent une dépendance des utilisateur-rices 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-rices de collaborer en l'absence d'autorités centrales pour résoudre les conflits de modifications ?

(ii) Comment authentifier les utilisateur-rices 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

État de l'art

Sommaire

1.0.1 Approche à pierres tombales

1.0.2 Approche à identifiants densément ordonnés

Treedoc

— [12, 13] proposent une nouvelle approche pour CRDTs pour Séquence. Se base sur des identifiants de position, respectant les propriétés suivantes :

(i) Chaque élément se voit attribuer un identifiant.

(ii) Aucune paire d'éléments ne partage le même identifiant.

(iii) L'identifiant d'un élément est immuable.

(iv) Il existe un ordre total strict sur les identifiants, $<_{id}$, cohérent avec l'ordre des éléments dans la séquence.

(v) Les identifiants sont tirés d'un ensemble dense, noté $\mathbb{I}$.

— La propriété (v) signifie que :

$$\forall predId, succId \in \mathbb{I}, \exists id \in \mathbb{I} \cdot predId <_{id} id <_{id} succId$$

Par exemple, les nombres réels forment un ensemble dense. Ceux-ci sont néanmoins inutilisables en informatique puisqu'ils nécessiteraient une précision infinie. [13] utilise donc un type dédié pour les émuler.

— L'utilisation d'identifiants de position permet de redéfinir les modifications :

(i) $ins(pred \prec elt \prec succ)$ devient alors $ins(id, elt)$, avec $predId <_{id} id <_{id} succId$.

(ii) $rmv(elt)$ devient $rmv(id)$.

Ces redéfinitions permettent d'obtenir une spécification de la séquence avec des modifications commutatives.

— À partir de cette spécification, PREGUICA et al. propose un CRDT pour Séquence : *Treedoc*. Ce dernier se base sur un arbre binaire pour générer les identifiants de position.

— La racine de l'arbre binaire, notée ϵ, correspond à l'identifiant du premier élément inséré dans la séquence répliquée. Puis, pour générer l'identifiant d'un nouvel élément inséré à gauche (resp. à droite) d'un noeud de l'arbre binaire, Treedoc concatène un 0 (resp. un 1) à l'identifiant de ce dernier. *Matthieu: TODO : Figure arbre binaire d'identifiants.*

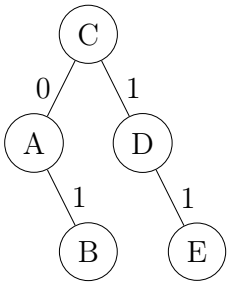

FIGURE 1.1 – Identifiants de positions

— Ce mécanisme souffre néanmoins d'un écueil : en l'état, plusieurs noeuds du systèmes peuvent générer un même identifiant pour des éléments insérés en concurrence, contravenant alors à la propriété (ii). Pour corriger cela, Treedoc ajoute aux identifiants des désambiguateurs : un *Dot*. Un désambiguateur est ajouté à chaque partie d'un identifiant lorsque nécessaire, c.-à-d.

(i) La partie courante de l'identifiant est la fin de l'identifiant.

(ii) La partie courante de l'identifiant nécessite désambiguation, c.-à-d. plusieurs éléments utilisent ce même identifiant.

Il convient de noter que les noeuds de l'arbre binaire des identifiants peuvent ainsi contenir une liste d'identifiants en cas d'insertions concurrentes. *Matthieu: TODO : Figure arbre binaire d'identifiants avec désambiguateurs et mini-nodes.*

— *Matthieu: TODO : Figure exemple d'édition collaborative*

— Concernant le modèle de livraison utilisé, [13] indique reposer sur le modèle de livraison causal. En pratique, nous pouvons néanmoins relaxer le modèle de livraison comme expliqué dans [5] :

(i) Les opérations *ins* peuvent être livrées dans n'importe quel ordre.

(ii) L'opération $rmv(id)$ ne peut délivrée qu'après la livraison de l'opération d'insertion de l'élément associé à id.

— Treedoc souffre néanmoins de plusieurs limites. Tout d'abord, le mécanisme d'identifiants de positions proposé est couplé à la structure d'arbre binaire. Cependant,

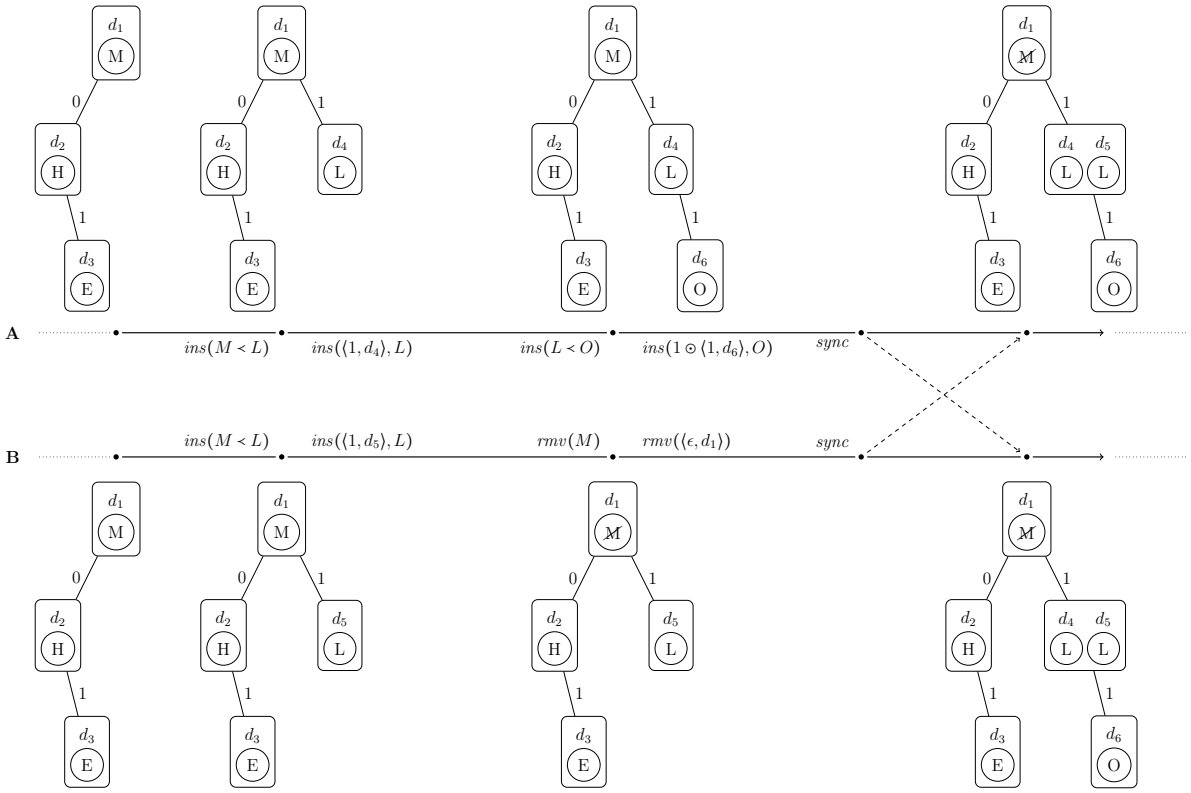

FIGURE 1.2 – Modifications concurrentes d'une séquence répliquée Treedoc

les utilisateur-rices ont tendance à écrire de manière séquentielle, c.-à-d. dans le sens d'écriture de la langue utilisée. Les nouveaux identifiants forment donc généralement une liste chaînée, qui déséquilibre l'arbre.

— Ensuite, Treedoc doit conserver un noeud de l'arbre des identifiants malgré sa suppression lorsque ce dernier possède des enfants. Ce noeud de l'arbre devient alors une pierre tombale. Le mécanisme de Garbage Collection (GC) des pierres tombales de Treedoc ne repose cependant pas sur la stabilité causale d'opérations, mais sur l'état du noeud de l'arbre, c.-à-d. si la pierre tombale devient une feuille. Néanmoins, l'évaluation de [13] a montré que les pierres tombales pouvait représenter jusqu'à 95% des noeuds de l'arbre.

— Finalement, Treedoc souffre du problème de l'entrelacement d'éléments insérés de manière concurrente, contrairement à ce qui est conjecturé dans [14].

Logoot

— Logoot [15, 16]

Matthieu: NOTE : Ajouter LogootSplit de manière sommaire aussi à cet endroit ?

Matthieu: TODO : Autres Sequence CRDTs à considérer : String-wise CRDT [17], Chronofold [18]

1.1 Mitigation du surcoût des séquences répliquées sans conflits

— Plusieurs approches ont été proposées pour réduire croissance des métadonnées dans Sequence CRDTs

— Replicated Growable Array (RGA) (et RGASplit) propose un mécanisme de GC des pierres tombales. Nécessite cependant stabilité causale des opérations de suppression. S'agit d'une contrainte forte, peu adaptée aux systèmes dynamiques à large échelle. *Matthieu: TODO : Trouver référence sur la stabilité causale dans systèmes dynamiques*

— Core & Nebula propose un mécanisme de ré-équilibrage de l'arbre pour Treedoc. Le ré-équilibrage a pour effet de supprimer des potentielles pierres tombales et de réduire la taille des identifiants. Repose sur un algorithme de consensus. S'agit de nouveau d'une contrainte forte pour systèmes dynamique à large échelle. Pour y pallier, propose de séparer les pairs entre deux ensembles : Core et Nebula. Permet de limiter le nombre participant au consensus. Un protocole de rattrapage permet aux noeuds de la Nebula de mettre à jour leurs modifications concurrentes à un ré-équilibrage.

— LSEQ adopte une autre approche. Part du constat que les identifiants dans Logoot croissent de manière linéaire. Vise une croissance logarithmique des identifiants. Pour cela, propose de nouvelles fonctions d'allocation des identifiants visant à maximiser le nombre d'identifiants insérés avant de devoir augmenter la taille de l'identifiant. Propose aussi d'utiliser une base exponentielle pour la valeur *position* des identifiants. Atteint ainsi la croissance polylogarithmique des identifiants, sans coordination requise entre les noeuds et mécanisme supplémentaire. Solution adaptée aux systèmes distribués à large échelle. Conjecture cependant que cette approche se marie mal avec les Sequence CRDTs utilisant des blocs. En effet, ajoute une raison supplémentaire à la croissance des identifiants : l'insertion entre identifiants contigus. Force alors la croissance des identifiants.

1.2 Synthèse

— Systèmes distribués adoptent le modèle de la réplication optimiste pour offrir de meilleures performances, c.-à-d. disponibilité et latence, et assurer la résilience du système, c.-à-d. accroître la capacité de tolérance aux pannes.

— Ce modèle autorise les noeuds à modifier leur copie sans coordination, provoquant ainsi des divergences temporaires. Pour résoudre les potentiels conflits et assurer la convergence à terme des copies, systèmes utilisent les CRDTs en place et lieu des types de données séquentiels.

— CRDTs pour Séquence ont été proposés pour conception d'éditeurs collaboratifs pair-à-pair. Deux approches sont utilisées pour concevoir leur mécanismes de résolution de conflits : l'approche basée sur les pierres tombales et l'approche basée sur les identifiants densément ordonnés.

— Chacune de ces approches introduit un surcoût croissant, pénalisant leurs performances à terme. Plusieurs travaux ont été proposés pour limiter ce surcoût, notamment [1, 2] qui présentent un mécanisme de renommage des identifiants pour les CRDTs pour Séquence basés sur identifiants densément ordonnés.

— Mais cette approche requiert un mécanisme de consensus, des renommages concurrents provoquant un nouveau conflit. Cette contrainte empêche son utilisation dans les systèmes pair-à-pair ne disposant pas de noeuds suffisamment stables et bien connectés pour exécuter le mécanisme de consensus.

1.3 Proposition

— Dans ce manuscrit, nous proposons et présentons un nouveau mécanisme de renommage pour CRDTs pour Séquence, ne nécessitant pas de coordination synchrone entre les noeuds.

— Concevons ce mécanisme pour le CRDT pour Séquence LogootSplit, mais principe de notre approche est générique. Pourrait ainsi l'adapter et proposer un équivalent pour autres CRDTs pour Séquence, e.g. RGASplit.

— Présentons et détaillons notre contribution dans le chapitre suivant.

Chapitre 2

Conclusions et perspectives

Sommaire

2.1 Résumé des contributions

2.2 Perspectives

2.2.1 Définition de relations de priorité pour minimiser les traitements

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

2.2.3 Définition de types de données répliquées sans conflits plus complexes

2.2.4 Étude comparative des différentes familles de CRDTs

— La spécification récente des Delta-based CRDTs . Ce nouveau type de CRDTs se base sur celui des State-based CRDTs. Partage donc les mêmes pré-requis :
  — États du type de données répliqué forment un sup-demi-treillis
  — Modifications locales entraînent une inflation de l'état
  — Possède une fonction de `merge`, permettant de fusionner deux états S et S', et qui
    — Est associative, commutative et idempotente
    — Retourne S", la Least Upper Bound (LUB) de S et S' (c.-à-d. $\nexists S''' \cdot merge(S, S') < S''' < S''$)

  Et bénéficie de son principal avantage : synchronisation possible entre deux pairs en fusionnant leur états, peu importe le nombre de modifications les séparant.

— Spécificité des Delta-based CRDTs est de proposer une synchronisation par différence d'états. Plutôt que de diffuser l'entièreté de l'état pour permettre aux autres pairs de se mettre à jour, idée est de seulement transmettre la partie de l'état ayant été mise à jour. Correspond à un élément irréductible du sup-demi-treillis. Permet ainsi de mettre en place une synchronisation en temps réel de manière efficace. Et d'utiliser la synchronisation par fusion d'états complets pour compenser les défaillances du réseau

— Ainsi, ce nouveau type de CRDTs semble allier le meilleur des deux mondes :
  — Absence de contrainte sur le réseau autre que la livraison à terme
  — Propagation possible en temps réel des modifications

  Semble donc être une solution universelle :
  — Utilisable peu importe la fiabilité réseau à disposition
  — Empreinte réseau du même ordre de grandeur qu'un Op-based CRDT
  — Utilisable peu importe la fréquence de synchronisation désirée

  Pose la question de l'intérêt des autres types de CRDTs.

— Delta-based CRDT est un State-based CRDT dont on a identifié les éléments irréductibles et qui utilise ces derniers pour la propagation des modifications plutôt que l'état complet. Famille des State-based CRDTs semble donc rendue obsolète par celle des Delta-based CRDTs. À confirmer.

— Les Op-based CRDTs proposent une spécification différente du type répliqué de leur équivalent Delta-based, généralement plus simple. À première vue, famille des Op-based CRDTs semble donc avoir la simplicité comme avantage par rapport à celle des Delta-based CRDTs. S'agit d'un paramètre difficilement mesurable et auquel on peut objecter si on considère qu'un Op-based CRDT s'accompagne d'une couche livraison de messages, qui cache sa part de complexité. Intéressant d'étudier si la spécification différente des Op-based CRDTs présente d'autres avantages par rapport aux Delta-based CRDTs : performances (temps d'intégration des modifications, délai de convergence...), fonctionnalités spécifiques (composition, undo...)

— But serait de fournir des guidelines sur la famille de CRDT à adopter en fonction du cas d'utilisation.

2.2.5 Définition d'opérations supplémentaires pour fonctionnalités liées à l'édition collaborative

— Commentaires

— Suggestions

2.2.6 Conduction d'expériences utilisateurs d'édition collaborative

— Absence d'un dataset réel et réutilisable sur les sessions d'édition collaborative

— Généralement, expériences utilisent données d'articles de Wikipédia *Matthieu: TODO : Revoir références, mais me semble que c'est celui utilisé pour Logoot, LogootSplit et RGASplit entre autres*. Mais ces données correspondent à une exécution séquentielle, c.-à-d. aucune édition concurrente ne peut être réalisée avec le système de résolution de conflits de Wikipédia. *Matthieu: TODO : Me semble que Kleppmann a aussi utilisé et mis à disposition ses traces correspondant à la rédaction d'un de ses articles. Mais que cet article n'était rédigé que par lui. Peu de chances de présence d'éditions concurrentes. À retrouver et vérifier.*

— Inspiré par expériences de Claudia, pourrait mener des sessions d'édition collaborative sur des outils orchestrés pour produire ce dataset

— Devrait rendre ce dataset agnostique de l'approche choisie pour la résolution automatique de conflits

— Absence de retours sur les collaborations à grande échelle

— Comment on collabore lorsque plusieurs centaines d'utilisateur-rices ?

2.2.7 Comparaison des mécanismes de synchronisation

Serait intéressant de comparer à d'autres méthodes de synchronisation : mécanisme d'anti-entropie basé sur un Merkle Tree[19, 20, 21], synchronisation par états (state/delta-based CRDTs). Dans le cadre des Delta-based CRDTs, pourrait évaluer un protocole de diffusion épidémique des deltas comme celui proposé par SWIM[8].

2.2.8 Distance entre versions d'un document

- Est-ce que ça a vraiment du sens d'intégrer automatiquement des modifications ayant été généré sur une version du document distante de l'état actuel du document (voir distance de Hamming, Levenstein, 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 ?*

2.2.9 Contrôle d'accès

- Pour le moment, n'importe quel utilisateur ayant l'URL du document peut y accéder dans MUTE
- Pour des raisons de confidentialité, peut vouloir contrôler quels utilisateurs ont accès à un document
- Nécessite l'implémentation de liste de contrôle d'accès
- Mais s'agit d'une tâche complexe dans le cadre d'un système distribué
- Peut s'inspirer des travaux réalisés au sein de la communautée CRDTs [22, 23] pour cela

2.2.10 Détection et éviction de pairs malhonnêtes

- À l'heure actuelle, MUTE suppose qu'ensemble des collaborateurs honnêtes
- Vulnérable à plusieurs types d'attaques par des adversaires byzantins, tel que l'équivoque
- Ce type d'attaque peut provoquer des divergences durables et faire échouer des collaborations
- Dans [24, 5], Elvinger propose un mécanisme permettant de maintenir des logs authentifiés dans un système distribué
- Les logs authentifiés permettent de mettre en lumière les comportements malveillants des adversaires et de borner le nombre d'actions malveillantes qu'ils peuvent effectuer avant d'être évincé
- Implémenter ce mécanisme permettrait de rendre compatible MUTE avec des environnements avec adversaires byzantins

— Nécessiterait tout de même de faire évoluer le CRDT pour résoudre les équivoques détectés

2.2.11 Vecteur *epoch-based*

— Comme présenté précédemment, nous utilisons plusieurs vecteurs pour représenter des données dans l'application MUTE

— Notamment pour le vecteur de version, utilisé pour respecter le modèle de livraison requis par le CRDT

— Et pour la liste des collaborateurs, utilisé pour offrir des informations nécessaires à la conscience de groupe aux utilisateurs

— Ces vecteurs sont maintenus localement par chacun des noeuds et sont échangés de manière périodique

— Cependant, la taille de ces vecteurs croit de manière linéaire au nombre de noeuds impliqués dans la collaboration

— Les systèmes Pair-à-Pair (P2P) à large échelle sont sujets au *churn*

— Dans le cadre d'un tel système, ces structures croissent de manière non-bornée

— Ceci pose un problème de performances, notamment d'un point de vue consommation en bande-passante

— Cependant, même si on observe un grand nombre de pairs différents dans le cadre d'une collaboration à large échelle

— Intuition est qu'une collaboration repose en fait sur un petit noyau de collaborateurs principaux

— Et que majorité des collaborateurs se connectent de manière éphèmère

— Serait intéressant de pouvoir réduire la taille des vecteurs en oubliant les collaborateurs éphèmères

— Dynamo[19] tronque le vecteur de version lorsqu'il dépasse une taille seuil

— Conduit alors à une perte d'informations

— Pour la liste des collaborateurs, approche peut être adoptée (pas forcément gênant de limiter à 100 la taille de la liste)

— Mais pour vecteur de version, conduirait à une relivraison d'opérations déjà observées

— Approche donc pas applicable pour cette partie

— Autre approche possible est de réutiliser le système d'époque

— Idée serait de ACK un vecteur avec un changement d'époque

— Et de ne diffuser à partir de là que les différences

— Un mécanisme de transformation (une simple soustraction) permettrait d'obtenir le dot dans la nouvelle époque d'une opération concurrente au renommage

— Peut facilement mettre en place un mécanisme d'inversion du renommage (une simple addition) pour revenir à une époque précédente

— Et ainsi pouvoir circuler librement dans l'arbre des époques et gérer les opérations *rename* concurrentes

— Serait intéressant d'étudier si on peut aller plus loin dans le cadre de cette structure de données et notamment rendre commutatives les opérations de renommage concurrentes

2.2.12 Fusion de versions distantes d'un document collaboratif

2.2.13 Rôles et places des bots dans systèmes collaboratifs

— Stockage du document pour améliorer sa disponibilité

— Overleaf en P2P ?

— Comment réinsérer des bots dans la collaboration sans en faire des éléments centraux, sans créer des failles de confidentialité, et tout en rendant ces fonctionnalités accessibles ?

Annexe A

Algorithmes RENAMEID

Algorithme 1 Remaining functions to rename an identifier

```
function renIdLessThanFirstId(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 B

Algorithmes revertRenameId

Algorithme 2 Remaining functions to revert an identifier renaming

```
function REVRENIDLESSTHANNEWFIRSTID(id, firstId, newFirstId)
    predNewFirstId ← createIdFromBase(newFirstId, -1)
    if isPrefix(predNewFirstId, id) then
        tail ← getTail(id, 1)
        if tail < firstId then
            return tail
        else
                                                ▷ id has been inserted causally after the rename op
            offset ← getLastOffset(firstId)
            predFirstId ← 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 ← 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

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 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.

[4] 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*. 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 et al. « 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] A. Das, I. Gupta et A. 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 et al. « 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.

[12] Marc Shapiro et Nuno Preguiça. *Designing a commutative replicated data type*. Research Report RR-6320. INRIA, 2007. URL : https://hal.inria.fr/inria-00177693.

[13] 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.

[14] Martin Kleppmann et al. « Interleaving Anomalies in Collaborative Text Editors ». In : *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '19. Dresden, Germany : Association for Computing Machinery, 2019. ISBN : 9781450362764. DOI : 10.1145/3301419.3323972. URL : https://doi.org/10.1145/3301419.3323972.

[15] 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.

[16] 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.

[17] Weihai Yu. « A String-Wise CRDT for Group Editing ». In : *Proceedings of the 17th ACM International Conference on Supporting Group Work*. GROUP '12. Sanibel Island, Florida, USA : Association for Computing Machinery, 2012, p. 141–144. ISBN : 9781450314862. DOI : 10.1145/2389176.2389198. URL : https://doi.org/10.1145/2389176.2389198.

[18] Victor GRISHCHENKO et Mikhail PATRAKEEV. « Chronofold : A Data Structure for Versioned Text ». In : *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '20. Heraklion, Greece : Association for Computing Machinery, 2020. ISBN : 9781450375245. DOI : 10.1145/3380787.3393680. URL : https://doi.org/10.1145/3380787.3393680.

[19] Giuseppe DECANDIA et al. « Dynamo : Amazon's highly available key-value store ». In : *ACM SIGOPS operating systems review* 41.6 (2007), p. 205–220.

[20] Nico KRUBER, Maik LANGE et Florian SCHINTKE. « Approximate Hash-Based Set Reconciliation for Distributed Replica Repair ». In : *2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS)*. 2015, p. 166–175. DOI : 10.1109/SRDS.2015.30.

[21] Ricardo Jorge Tomé GONÇALVES et al. « DottedDB : Anti-Entropy without Merkle Trees, Deletes without Tombstones ». In : *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*. 2017, p. 194–203. DOI : 10.1109/SRDS.2017.28.

[22] Elena YANAKIEVA et al. « Access Control Conflict Resolution in Distributed File Systems Using CRDTs ». In : *Proceedings of the 8th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '21. Online, United Kingdom : Association for Computing Machinery, 2021. ISBN : 9781450383387. DOI : 10.1145/3447865.3457970. URL : https://doi.org/10.1145/3447865.3457970.

[23] Pierre-Antoine RAULT, Claudia-Lavinia IGNAT et Olivier PERRIN. « Distributed Access Control for Collaborative Applications Using CRDTs ». In : *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*. PaPoC '22. Rennes, France : Association for Computing Machinery, 2022, p. 33–38. ISBN : 9781450392563. DOI : 10.1145/3517209.3524826. URL : https://doi.org/10.1145/3517209.3524826.

[24] Victorien ELVINGER, Gérald OSTER et François CHAROY. « Prunable Authenticated Log and Authenticable Snapshot in Distributed Collaborative Systems ». In : *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2018, p. 156–165.

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