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

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

Mis en page avec la classe thesul.

Remerciements

WIP

WIP

Sommaire

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Contexte

L'évolution des technologies du web a conduit à l'avènement de ce qui est communément appelé le Web 2.0. La principale caractéristique de ce média est la possibilité aux utilisateur-rices non plus seulement de le consulter, mais aussi d'y contribuer.

Ces nouvelles fonctionnalités ont permis l'apparition d'applications incitant les utilisateur-rices à créer et partager leur propre contenu, ainsi que d'échanger avec d'autres utilisateur-rices à ce sujet. [1] définit ce type d'applications, c.-à-d. les *réseaux sociaux*, de la manière suivante :

Définition 1. Un réseau social est une application respectant les critères suivants :

(i) Elle prend la forme d'une application interactive.

(ii) Elle permet à ses utilisateur-rices de produire et de partager leur propre contenu. Ce contenu généré par les utilisateur-rices constitue le contenu principal de l'application.

(iii) Elle permet à ses utilisateur-rices de possèder et de maintenir leur propre profil sur la plateforme.

(iv) Elle encourage ses utilisateur-rices à étendre leur réseau social en se connectant à d'autres utilisateur-rices, voire en créant ou en s'intégrant à des communautés.

De nos jours, les réseaux sociaux représentent les applications les plus populaires du paysage internet, e.g. Facebook compte 2,9 millards d'utilisateur-rices par mois [2], Youtube 2,5 milliards [2], Wikipedia 788 millions [3] ou encore Quora 300 millions [2].

La démocratisation de ces applications présente plusieurs bienfaits. Notamment, nous notons que les réseaux sociaux permettent, en diminuant le coût individuel pour tout à chacun-e de contribuer, d'améliorer la diffusion de l'information [4, 5]. Aussi, ils contribuent à la création de communautés [6]. Finalement, en permettant à chacun-e de partager son savoir, ils permettent la création de bases de connaissances complètes [7, 8].

Cependant, la conception de réseaux sociaux fait face à de nombreux défis. Notamment, ces applications doivent assurer leur haute disponibilité, tolérance aux pannes et capacité de passage à l'échelle en raison de leur popularité et importance dans notre quotidien.

Définition 2 (Disponibilité)**.** La disponibilité d'un système indique sa capacité à répondre à tout moment à une requête d'un-e utilisateur-rice.

Définition 3 (Tolérance aux pannes)**.** La tolérance aux pannes d'un système indique sa capacité à continuer à répondre aux requêtes malgré l'absence de réponse d'un ou plusieurs de ses composants.

Définition 4 (Capacité de passage à l'échelle)**.** La capacité de passage à l'échelle d'un système indique sa capacité à traiter un volume toujours plus conséquent de requêtes.

Pour cela, ces systèmes adoptent une architecture décentralisée [1], c.-à-d. une architecture reposant sur un ensemble de serveurs qui se répartissent la charge de travail et les tâches. Malgré ce que le nom de cette architecture peut suggérer, il convient de noter que de manière globale les serveurs jouent toujours un rôle central dans les systèmes décentralisés. Par exemple, les serveurs servent à authentifier les utilisateur-rices, à stocker leurs données ou encore à assurer la communication entre utilisateur-rices.

Additionnellement, il convient de préciser que ces serveurs ne sont pas une ressource libre. En effet, ils sont mis à disposition et maintenus par la ou les organisations qui proposent le réseau social. Ces organisations font alors office d'*autorités centrales* du système, e.g. en se portant garantes de l'identité des utilisateur-rices ou encore de l'authenticité d'un contenu.

De part le rôle que jouent les serveurs dans les systèmes décentralisés, ces derniers échouent à assurer un second ensemble de propriétés, que nous jugeons néanmoins fondamentales :

Définition 5 (Confidentialité des données)**.** La confidentialité des données d'un système indique sa capacité à garantir à ses utilisateur-rices que leurs données ne seront pas accessibles par des tiers non autorisés ou par le système lui-même.

Définition 6 (Souveraineté des données)**.** La souveraineté des données d'un système indique sa capacité à garantir à ses utilisateur-rices leur maîtrise de leurs données, c.-à-d. leur capacité à les consulter, modifier, partager, exporter ; supprimer ou encore à décider de l'usage qui en est fait.

1. Nous utilisons la classification présentée dans [9] pour distinguer les architectures systèmes centralisées, décentralisées et distribuées.

Définition 7 (Pérennité)**.** La pérennité d'un système indique sa capacité à garantir à ses utilisateur-rices son fonctionnement continu dans le temps.

Définition 8 (Résistance à la censure)**.** La résistance à la censure d'un système indique sa capacité à garantir à ses utilisateur-rices son fonctionnement malgré des actions de contrôle de l'information par des autorités.

Ainsi, les utilisateur-rices des réseaux sociaux prennent, de manière consciente ou non, le risque que ces propriétés soient transgressées par les autorités auxquelles appartiennent ces applications ou par des tiers avec lesquelles ces autorités interagissent, e.g. des gouvernements.

Qui plus est, il convient de noter que les autorités auxquelles appartiennent ces applications, e.g. des entreprises, possèdent leurs propres intérêts, e.g. leur propre profit. De nombreux faits d'actualité ont malheureusement montré que ces autorités ont tendance à privilégier leurs intérêts, quitte à prendre des décisions s'avérant nocives pour une partie de leurs utilisateur-rices. Nous pouvons par exemple noter la mise en avant de contenu à caractère raciste [10], la non-modération de contenu misogyne [11], l'encouragement à la production de contenu toxique [12] ou l'inaction et même entrave à la correction des effets délétères de son réseau social sur la santé mentale de ses utilisatrices [13].

Ainsi, la présence d'autorités centrales dans les réseaux sociaux représente un danger pour les utilisateur-rices. Il nous paraît alors fondamental de proposer des moyens technologiques pour concevoir et déployer des réseaux sociaux alternatifs qui minimisent le rôle des autorités centrales, voire l'éliminent.

Dans cette optique, une piste de recherche que nous jugeons intéressante à étudier est celle présentée dans [14]. Dans ce travail, les auteurs proposent un nouveau paradigme de conception d'applications, nommées Local-First Softwares (LFS). Ce type d'applications se démarque de ceux existants, e.g. les applications basées sur le cloud, par la place centrale donnée aux utilisateur-rices et leurs propres appareils, les éventuels serveurs étant relegués qu'à de simples rôles de support.

[14] identifie 7 propriétés que doivent satisfaire les applications LFS, illustrées dans la Figure 1.1. Ces propriétés se recoupent en grande partie avec celles que nous avons

	1. Fast	2. Multi-device	3. Offline	4. Collaboration	5. Longevity	6. Privacy	7. User control
???	✔	✔	✔	✔	✔	✔	✔

FIGURE 1.1 – Liste des propriétés visées par les LFS [2]

identifiées précédemment :

(i) Le fonctionnement en mode hors-ligne et le fonctionnement avec une latence minimale sont tous deux assurés en privilégiant la disponibilité [15] (Définition 2).

(ii) Le respect de la vie privée des utilisateur-rices correspond à la propriété de confidentialité (Définition 5).

2. Source : `https://www.inkandswitch.com/local-first/#towards-a-better-future`

(iii) Le contrôle des utilisateur-rices sur leurs données correspond à la propriété de souveraineté (Définition 6).

(iv) La longétivité de l'application correspond aux propriétés de pérennité (Définition 7) et de résistance à la censure (Définition 8).

Notons qu'il découle de ces propriétés que les applications LFS sont fondamentalement P2P.

De manière similaire, observons que ce paradigme met en lumière une dernière propriété des applications LFS :

Définition 9 (Collaborativité)**.** La collaborativité d'un système indique sa capacité à supporter ses utilisateur-rices dans leurs processus de collaboration pour la réalisation de tâches.

Matthieu: TODO : Introduire notion d'agents artificiels/logiciels Nous précisons que nous considérons dans ce manuscrit qu'une collaboration peut prendre bien des formes. Une collaboration peut ainsi prendre la forme d'un échange entre utilisateur-rices, e.g. un fil de discussion sur une plateforme de questions et réponses, ou d'une édition collaborative d'un contenu, e.g. la rédaction d'une page de wiki.

Matthieu: TODO : Voir si angle écologique/réduction consommation d'énergie peut être pertinent.

Ainsi, [14] établit un paradigme de conception d'applications correspondant à notre vision. Cependant, de nombreuses problématiques de recherche identifiées dans ce travail sont encore non résolues et entravent la démocratisation des applications LFS.

Notamment, les applications LFS se doivent de répliquer les données entre appareils pour permettre :

(i) Le fonctionnement en mode hors-ligne et le fonctionnement avec une faible latence.

(ii) Le partage de contenu entre appareils d'un-e même utilisateur-rice.

(iii) Le partage de contenu entre utilisateur-rices pour la collaboration.

Cependant, compte tenu des propriétés visées par les applications LFS, plusieurs contraintes restreignent le choix des méthodes de réplication possibles. Ainsi, pour permettre le fonctionnement en mode hors-ligne de l'application, c.-à-d. la consultation et la modification de contenu, les applications LFS doivent relaxer la propriété de cohérence des données.

Définition 10 (Cohérence)**.** La cohérence d'un système indique sa capacité à présenter une vue uniforme de son état à chacun de ses utilisateur-rices à un moment donné.

Les applications LFS ne peuvent donc pas reposer sur des méthodes de réplication dites pessimistes, c.-à-d. qui empêchent toutes modifications concurrentes d'une même donnée.

Les applications LFS doivent donc adopter des méthodes de réplication dites optimistes [16]. Ces méthodes autorisent chaque noeud possédant une copie de la donnée de la consulter et de la modifier sans coordination au préalable avec les autres noeuds. L'état

des copies des noeuds peut donc diverger temporairement. Un mécanisme de synchronisation permet ensuite aux noeuds de partager les modifications effectuées et de les intégrer de façon à converger à terme [17], c.-à-d. obtenir à terme de nouveau des états équivalents.

Cependant, il convient de noter que les méthodes de réplication optimistes autorisent la génération en concurrence de modifications provoquant un conflit, e.g. la modification et la suppression d'une même page dans un wiki. Un mécanisme de résolution de conflits est alors nécessaire pour assurer la convergence à terme des noeuds.

De nouveau, le modèle du système des applications LFS limitent les choix possibles concernant les mécanismes de résolution de conflits. Notamment, les applications LFS ne disposent d'aucun contrôle sur le nombre de noeuds qui compose le système, c.-à-d. le nombre d'appareils utilisés par l'ensemble de leurs utilisateur-rices. Le nombre de noeuds peut donc croître de manière non-bornée. La complexité algorithmique des mécanismes de résolution de conflits doit donc être indépendante de ce paramètre, ou alors en être fonction uniquement de manière logarithmique.

De plus, ces noeuds n'offrent aucune garantie sur leur stabilité. Des noeuds peuvent donc rejoindre et participer au système, mais uniquement de manière éphèmère. Ce phénonème est connu sous le nom de *churn* [18]. Ainsi, de part l'absence de garantie sur le nombre de noeuds connectés de manière stable, les applications LFS ne peuvent pas utiliser des mécanismes de résolution de conflits reposant sur une coordination synchrone d'une proportion des noeuds du système, c.-à-d. sur des algorithmes de consensus [19, 20].

Ainsi, pour permettre la conception d'applications LFS, il convient de disposer de mécanismes de résolution de conflits pour l'ensemble des types de données avec une complexité algorithmique efficace par rapport au nombre de noeuds et ne nécessitant pas de coordination synchrone entre une proportion des noeuds du système.

1.2 Questions de recherche et contributions

1.2.1 Ré-identification sans coordination synchrone pour les CRDTs pour le type Séquence

Les Conflict-free Replicated Data Types (CRDTs) [21, 22] sont des types de données répliqués. Ils sont conçus pour permettre à un ensemble de noeuds d'un système de répliquer une donnée et pour leur permettre de la consulter et dee la modifier sans aucune coordination préalable. Dans ce but, les CRDTs incorporent des mécanismes de résolution de conflits automatiques directement au sein leur spécification.

Cependant, ces mécanimes induisent un surcoût, aussi bien en termes de métadonnées et de calculs que de bande-passante. Ces surcoûts sont néanmoins jugés acceptables par la communauté pour une variété de types de données, e.g. le Registre ou l'Ensemble. Cependant, le surcoût des CRDTs pour le type Séquence constitue toujours une problématique de recherche.

En effet, la particuliarité des CRDTs pour le type Séquence est que leur surcoût croît de manière monotone au cours de la durée de vie de la donnée, c.-à-d. au fur et à mesure des modifications effectuées. Le surcoût introduit par les CRDTs pour ce type de données se révèle donc handicapant dans le contexte de collaborations sur de longues durées ou à

large échelle.

De manière plus précise, le surcoût des CRDTs pour le type Séquence provient de la croissance des métadonnées utilisées par leur mécanisme de résolution de conflits automatique. Ces métadonnées correspondent à des identifiants qui sont associés aux éléments de la Séquence. Ces identifiants permettent de résoudre les conflits, e.g. en précisant quel est l'élement à supprimer ou en spécifiant la position d'un nouvel élément à insérer par rapport aux autres.

Plusieurs approches ont été proposées pour réduire le coût induit par ces identifiants. Notamment, [23, 24] proposent un mécanisme de ré-assignation des identifiants pour réduire leur coût a posteriori. Ce mécanisme génère toutefois des conflits en cas de modifications concurrentes de la séquence, c.-à-d. l'insertion ou la suppression d'un élément. Les auteurs résolvent ce problème en proposant un mécanisme de transformation des modifications concurrentes par rapport à l'effet du mécanisme de ré-assignation des identifiants.

Cependant, l'exécution en concurrence du mécanisme de ré-assignation des identifiants par plusieurs noeuds provoque elle-même un conflit. Pour éviter ce dernier type de conflit, les auteurs choisissent de subordonner à un algorithme de consensus l'exécution du mécanisme de ré-assignation des identifiants. Ainsi, le mécanisme de ré-assignation des identifiants ne peut être déclenché en concurrence par plusieurs noeuds du systèmes.

Comme nous l'avons évoqué précédemment, reposer sur un algorithme de consensus qui requiert une coordination synchrone entre une proportion de noeuds du système est une contrainte incompatible avec les systèmes P2P à large échelle sujets au churn. Notre problématique de recherche est donc la suivante : *pouvons-nous proposer un mécanisme sans coordination synchrone de réduction du surcoût des CRDTs pour Séquence, c.-à-d. adapté aux applications LFS ?*

Pour répondre à cette problématiquee, nous proposons RenamableLogootSplit, un nouveau CRDT pour le type Séquence. Ce CRDT intègre un mécanisme de ré-assignation des identifiants, dit de renommage, directement au sein de sa spécification. Nous associons au mécanisme de renommage un mécanisme de résolution de conflits automatique additionnel pour gérer ses exécutions concurrentes. Ainsi, nous proposons un CRDT pour le type Séquence dont le surcoût est périodiquement réduit par le biais d'un mécanisme n'introduisant aucune contrainte de coordination synchrone entre les noeuds du système.

1.2.2 Éditeur de texte collaboratif P2P temps réel chiffré de bout en bout

Les systèmes collaboratifs permettent à plusieurs utilisateur-rices de collaborer pour la réalisation d'une tâche. Les systèmes collaboratifs actuels adoptent principalement une architecture décentralisée, c.-à-d. un ensemble de serveurs avec lesquels les utilisateur-rices interagissent pour réaliser leur tâche, e.g. Google Docs [25]. Par rapport à une architecture centralisée, cette architecture leur permet d'améliorer leur disponibilité et tolérance aux pannes, notamment grâce aux méthodes de réplication de données. Cette architecture à base de serveurs facilite aussi la collaboration, les serveurs permettant d'intégrer les modifications effectuées par les utilisateur-rices, de stocker les données, d'assurer la communication entre les utilisateur-rices ou encore de les authentifier.

De part le rôle qui leur incombe, ces serveurs occupent une place primordiale dans ces systèmes. Il en découle plusieurs problématiques :

(i) Ces serveurs manipulent et hébergent les données faisant l'objet de collaborations. Ces systèmes ont donc connaissance des données manipulées et de l'identité des auteur-rices des modifications. Les systèmes collaboratifs décentralisés demandent donc à leurs utilisateur-rices d'abandonner la souveraineté et la confidentialité de leur travail.

(ii) Ces serveurs sont gérés par des autorités centrales, e.g. Google. Les systèmes collaboratifs devenant non-fonctionnels en cas d'arrêt de leurs serveurs, les utilisateur-rices de ces systèmes dépendent de ces autorités centrales. Ainsi, de part leur pouvoir de vie et de mort sur les services qu'elles proposent, les autorités centrales représentent une menace pour la pérennité de ces systèmes, e.g. [26].

Pour répondre à ces problématiques, c.-à-d. confidentialité et souveraineté des données, dépendance envers des tiers, pérennité des systèmes, un nouveau paradigme de conception d'applications propose de concevoir des applications LFS, c.-à-d. des applications mettant les utilisateur-rices et leurs appareils au coeur du système. Dans ce cadre d'applications P2P, les serveurs sont relégués seulement à un rôle de support à la collaboration.

Dans le cadre de ses travaux, notre équipe de recherche étudie notamment la conception d'applications respectant ce paradigme. Ce changement de modèle, d'une architecture décentralisée appartenant à des autorités centrales à une architecture P2P sans autorités centrales, 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 connexions par pair ?

Cet ensemble de questions peut être résumé en la problématique suivante : *pouvons-nous concevoir une application collaborative P2P à large échelle, sûre et sans autorités centrales ?*

Pour étudier cette problématique, l'équipe Coast développe l'application Multi User Text Editor (MUTE) [3] [27]. Il s'agit d'un éditeur de texte web collaboratif P2P temps réel chiffré de bout en bout. Ce projet nous permet de présenter les travaux de recherche de l'équipe portant sur les mécanismes de résolutions de conflits automatiques pour le type Séquence [28, 29, 30] et les mécanismes d'authentification des pairs dans les systèmes sans autorités centrales [31, 32]. Puis, ce projet nous donne l'opportunité d'étudier la littérature des nombreux domaines de recherche nécessaires à la conception d'un tel système, c.-à-d. le domaine des protocoles d'appartenance aux groupes [33, 34], des topologies réseaux P2P [35] ou encore des protocoles d'établissement de clés de chiffrement de groupe [36]. Ce projet nous permet ainsi de valoriser nos travaux et d'identifier de nouvelles perspectives de recherche. Finalement, il résulte de ce projet le Proof of Concept (PoC) le plus complet d'applications LFS, à notre connaissance. *Matthieu: TODO : Vérifier du côté des applis de IPFS*

3. Disponible à l'adresse : `https://mutehost.loria.fr`

1.3 Plan du manuscrit

Ce manuscrit de thèse est organisé de la manière suivante :

Dans le **??**, nous introduisons le modèle du système que nous considérons, c.-à-d. les systèmes P2P à large échelle sujets au churn et sans autorités centrales. Puis nous présentons dans ce chapitre l'état de l'art des mécanismes de résolution de conflits automatiques utilisés dans les systèmes adoptant le paradigme de la réplication optimiste. À partir de cet état de l'art, nous identifions et motivons notre problématique de recherche, c.-à-d. l'absence de mécanisme adapté aux systèmes P2P à large échelle sujets au churn permettant de réduire le surcoût induit par les mécanismes de résolution de conflits automatiques pour le type Séquence.

Dans le **??**, nous présentons notre approche pour présenter un tel mécanisme, c.-à-d. un mécanisme de résolution de conflits automatiques pour le type Séquence auquel nous associons un mécanisme de Garbage Collection (GC) de son surcoût ne nécessitant pas de coordination synchrone entre les noeuds du système. Nous détaillons le fonctionnement de notre approche, sa validation par le biais d'une évaluation empirique puis comparons notre approche par rapport aux approches existantes Finalement, nous concluons la présentation de notre approche en identifiant et en détaillant plusieurs de ses limites.

Dans le chapitre 2, nous présentons MUTE, l'éditeur de texte collaboratif temps réel P2P chiffré de bout en bout que notre équipe de recherche développe dans le cadre de ses travaux de recherche. Nous présentons les différentes couches logicielles formant un pair et les services tiers avec lesquels les pairs interagissent, et détaillons nos travaux dans le cadre de ce projet, c.-à-d. l'intégration de notre mécanisme de résolution de conflits automatiques pour le type Séquence et le développement de la couche de livraison des messages associée. Pour chaque couche logicielle, nous identifions ses limites et présentons de potentielles pistes d'améliorations.

Finalement, nous récapitulons dans le chapitre 3 les contributions réalisées dans le cadre de cette thèse. Puis nous clotûrons ce manuscrit en introduisant plusieurs des pistes de recherches que nous souhaiterons explorer dans le cadre de nos travaux futurs.

1.4 Publications

Notre travail sur la problématique identifiée dans la sous-section 1.2.1, c.-à-d. la proposition d'un mécanisme ne nécessitant aucune coordination synchrone pour réduire le surcoût des CRDTs pour le type Séquence, a donné lieu à des publications à différents stades de son avancement :

(i) Dans [37], nous motivons le problème identifié et présentons l'idée de notre approche pour y répondre.

(ii) Dans [38], nous détaillons une première partie de notre approche et présentons notre protocole d'évaluation expérimentale ainsi que ses premiers résultats.

(iii) Dans [30], nous détaillons notre proposition dans son entièreté. Nous accompagnons cette proposition d'une évaluation expérimentale poussée. Finalement, nous complétons notre travail d'une discussion identifiant plusieurs de ses limites et présentant des pistes de travail possibles pour y répondre.

Nous précisons ci-dessous les informations relatives à chacun de ces articles.

Efficient renaming in CRDTs [37]

Auteur Matthieu Nicolas

Article de position à Middleware 2018 - 19th ACM/IFIP International Middleware Conference (Doctoral Symposium), Dec 2018, Rennes, France.

Abstract *Sequence Conflict-free Replicated Data Types (CRDTs)* allow to replicate and edit, without any kind of coordination, sequences in distributed systems. To ensure convergence, existing works from the literature add metadata to each element but they do not bound its footprint, which impedes their adoption. Several approaches were proposed to address this issue but they do not fit a fully distributed setting. In this paper, we present our ongoing work on the design and validation of a fully distributed renaming mechanism, setting a bound to the metadata's footprint. Addressing this issue opens new perspectives of adoption of these CRDTs in distributed applications.

Efficient Renaming in Sequence CRDTs [38]

Auteurs Matthieu Nicolas, Gérald Oster, Olivier Perrin

Article de workshop à PaPoC 2020 - 7th Workshop on Principles and Practice of Consistency for Distributed Data, Apr 2020, Heraklion / Virtual, Greece.

Abstract To achieve high availability, large-scale distributed systems have to replicate data and to minimise coordination between nodes. Literature and industry increasingly adopt Conflict-free Replicated Data Types (CRDTs) to design such systems. CRDTs are data types which behave as traditional ones, e.g. the Set or the Sequence. However, unlike traditional data types, they are designed to natively support concurrent modifications. To this end, they embed in their specification a conflict-resolution mechanism.

To resolve conflicts in a deterministic manner, CRDTs usually attach identifiers to elements stored in the data structure. Identifiers have to comply with several constraints, such as uniqueness or belonging to a dense order. These constraints may hinder the identifiers' size from being bounded. As the system progresses, identifiers tend to grow. This inflation deepens the overhead of the CRDT over time, leading to performance issues.

To address this issue, we propose a new CRDT for Sequence which embeds a renaming mechanism. It enables nodes to reassign shorter identifiers to elements in an uncoordinated manner. Experimental results demonstrate that this mechanism decreases the overhead of the replicated data structure and eventually limits it.

Efficient Renaming in Sequence CRDTs [30]

Auteurs Matthieu Nicolas, Gérald Oster, Olivier Perrin

Article de journal dans IEEE Transactions on Parallel and Distributed Systems, Institute of Electrical and Electronics Engineers, 2022, 33 (12), pp.3870-3885.

Abstract To achieve high availability, large-scale distributed systems have to replicate data and to minimise coordination between nodes. For these purposes, literature and industry increasingly adopt Conflict-free Replicated Data Types (CRDTs) to design such systems. CRDTs are new specifications of existing data types, e.g. Set or Sequence. While CRDTs have the same behaviour as previous specifications in sequential executions, they actually shine in distributed settings as they natively support concurrent updates. To this end, CRDTs embed in their specification conflict resolution mechanisms. These mechanisms usually rely on identifiers attached to elements of the data structure to resolve conflicts in a deterministic and coordination-free manner. Identifiers have to comply with several constraints, such as being unique or belonging to a dense total order. These constraints may hinder the identifier size from being bounded. Identifiers hence tend to grow as the system progresses, which increases the overhead of CRDTs over time and leads to performance issues. To address this issue, we propose a novel Sequence CRDT which embeds a renaming mechanism. It enables nodes to reassign shorter identifiers to elements in an uncoordinated manner. Experimental results demonstrate that this mechanism decreases the overhead of the replicated data structure and eventually minimises it.

Chapitre 2

MUTE, un éditeur de texte web collaboratif P2P temps réel chiffré de bout en bout

Sommaire

Les systèmes collaboratifs temps réels permettent à plusieurs utilisateur-rices de réaliser une tâche de manière coopérative. Ils permettent aux utilisateur-rices de consulter le contenu actuel, de le modifier et d'observer en direct les modifications effectuées par les autres collaborateur-rices. L'observation en temps réel des modifications des autres favorise une réflexion de groupe et permet une répartition efficace des tâches. L'utilisation des systèmes collaboratifs se traduit alors par une augmentation de la qualité du résultat produit [39, 7].

Plusieurs outils d'édition collaborative centralisés basés sur l'approche Operational Transformation (OT) [40] ont permis de populariser l'édition collaborative temps réel

de texte [25, 41]. Ces approchent souffrent néanmoins de leur architecture centralisée. Notamment, ces solutions rencontrent des difficultés à passer à l'échelle [42, 43] et posent des problèmes de confidentialité [44, 45].

L'approche CRDT offre une meilleure capacité de passage à l'échelle et est compatible avec une architecture P2P [46]. Ainsi, de nombreux travaux [47, 48, 49] ont été entrepris pour proposer une alternative distribuée répondant aux limites des éditeurs collaboratifs centralisés. De manière plus globale, ces travaux s'inscrivent dans le nouveau paradigme d'application des *Local-First Softwares* [14, 50]. Ce paradigme vise le développement d'applications collaboratives, P2P, pérennes et rendant la souveraineté de leurs données aux utilisateurs.

Matthieu: TODO : Serait intéressant d'ajouter une catégorisation des éditeurs collaboratifs en fonction de leurs caractéristiques (décentralisé vs. p2p, pas de chiffrement vs. chiffrement serveur vs. chiffrement de bout en bout, OT vs CRDT vs mécanisme de résolution de conflits custom...) pour mettre en avant le caractère unique de MUTE

De manière semblable, l'équipe Coast conçoit depuis plusieurs années des applications avec ces mêmes objectifs et étudient les problématiques de recherche liées. Elle développe Multi User Text Editor (MUTE) [27] [4] [5], un éditeur collaboratif P2P temps réel chiffré de bout en bout. MUTE sert de plateforme d'expérimentation et de démonstration pour les travaux de l'équipe.

Ainsi, nous avons contribué à son développement dans le cadre de cette thèse. Notamment, nous avons participé à :

(i) L'implémentation des CRDTs LogootSplit [28] et RenamableLogootSplit [30] pour représenter le document texte.

(ii) L'implémentation de leur modèle de livraison de livraison respectifs.

(iii) L'implémentation d'un protocole d'appartenance au réseau, SWIM [33].

Dans ce chapitre, nous commençons par présenter le projet MUTE : ses objectifs, ses fonctionnalités et son architecture système et logicielle. Puis nous détaillons ses différentes couches logicielles : leur rôle, l'approche choisie pour leur implémentation et finalement leurs limites actuelles. Au cours de cette description, nous mettons l'emphase sur les composants auxquelles nous avons contribué, c.-à-d. les sous-sections 2.3.1, 2.3.2 et 2.3.3.

2.1 Présentation

2.1.1 Objectifs

Comme indiqué dans l'introduction (cf. section 1.1, page 1), le but de ce projet est de proposer un éditeur de texte collaboratif Local-First Software (LFS), c.-à-d. un éditeur de texte collaboratif qui satisfait les propriétés suivantes :

4. Disponible à l'adresse : `https://mutehost.loria.fr`
5. Code source disponible à l'adresse suivante : `https://github.com/coast-team/mute`

(i) Toujours disponible, c.-à-d. qui permet à tout moment à un-e utilisateur-rice de consulter, créer ou éditer un document, même par exemple en l'absence de connexion internet.

(ii) Collaboratif, c.-à-d. qui permet à un-e utilisateur-rice de partager un document avec d'autres utilisateur-rices pour éditer à plusieurs le document, de manière synchrone et asynchrone. Nous considérons la capacité d'un-e utilisateur-rice à partager le document avec ses propres autres appareils comme un cas particulier de collaboration.

(iii) Performant, c.-à-d. qui garantit que le délai entre la génération d'une modification par un pair et l'intégration de cette dernière par un autre pair connecté soit assimilable à du temps réel et que ce délai ne soit pas impacté par le nombre de pairs dans la collaboration.

(iv) Pérenne, c.-à-d. qui garantit à ses utilisateur-rices qu'iels pourront continuer à utiliser l'application sur une longue période. Notamment, nous considérons la capacité des utilisateur-rices à configurer et déployer aisément leur propre instance du système comme un gage de pérennité du système.

(v) Garantissant la confidentialité des données, c.-à-d. qui permet à un-e utilisateur-rice de contrôler avec quelles personnes une version d'un document est partagée. Aussi, le système doit garantir qu'un adversaire ne doit pas être en mesure d'espionner les utilisateur-rices, e.g. en usurpant l'identité d'un-e utilisateur-rice ou en interceptant les messages diffusés sur le réseau.

(vi) Garantissant la souveraineté des données, c.-à-d. qui permet à un-e utilisateur-rice de maîtriser l'usage de ses données, e.g. pouvoir les consulter, modifier, partager ou encore exporter vers d'autres formats ou applications.

Ainsi, ces différentes propriétés nous conduisent à concevoir un éditeur de texte collaboratif P2P temps réel chiffré de bout en bout et qui est dépourvu d'autorités centrales.

2.1.2 Fonctionnalités

MUTE prend la forme d'une application web qui permet de créer et de gérer des documents textes. Chaque document se voit attribuer un identifiant, supposé unique. L'utilisateur-rice peut alors ouvrir et partager un document à partir de son URL.

L'application permet à l'utilisateur-rice d'être mis-e en relation avec les autres pairs actuellement connectés qui travaillent sur ce même document. Pour cela, l'application utilise le protocole WebRTC afin d'établir des connexions P2P avec ces derniers. Une fois les connexions P2P établies, le service fourni par le système pour mettre en relation les pairs n'est plus nécessaire.

Une fois connecté à un autre pair, l'utilisateur-rice récupère automatiquement les modifications effectuées par ses pairs de façon à obtenir la version courante du document. Iel peut alors modifier le document, c.-à-d. ajouter, supprimer du contenu ou encore modifier son titre. Ses modifications sont partagées en temps réel aux autres pairs connectés. À la réception de modifications, celles-ci sont intégrées à la copie locale du document. Figure 2.1 illustre l'interface utilisateur de l'éditeur de document de MUTE.

Pour garantir la confidentialité des échanges, MUTE utilise un protocole de génération de clés de groupe. Ce protocole permet d'établir une clé de chiffrement connue seulement

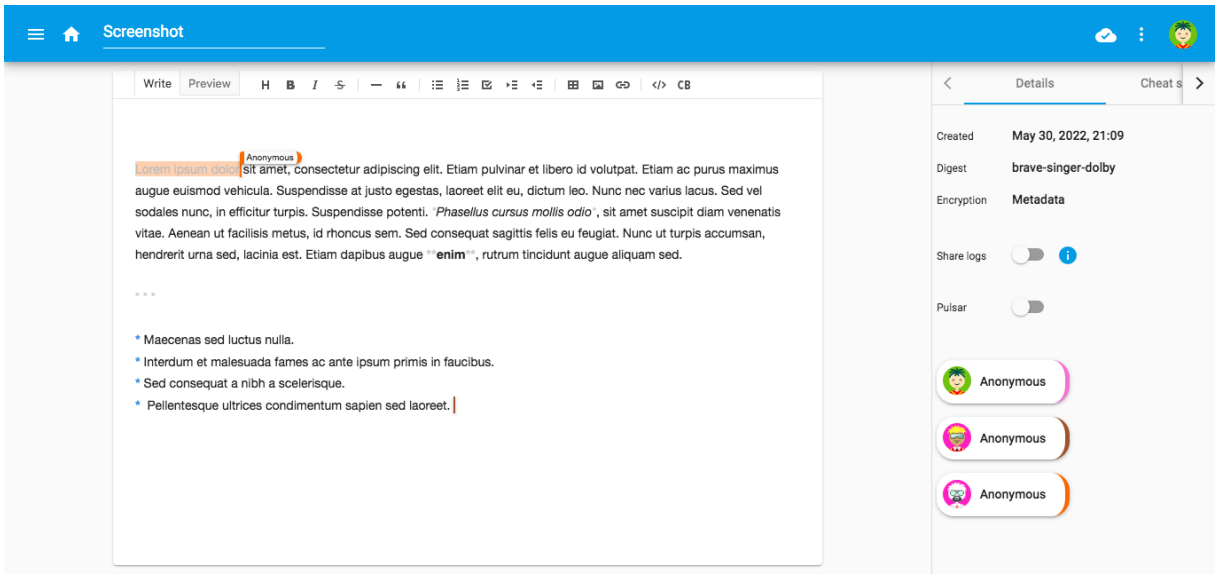


FIGURE 2.1 – Capture d'écran d'une session d'édition collaborative avec MUTE

des pairs actuellement connectés, qui est ensuite utilisée pour chiffrer les messages entre pairs. Ce protocole permet de garantir les propriétés de *backward secrecy* et de *forward secrecy*.

Définition 11 (Backward Secrecy)**.** La *Backward Secrecy* est une propriété de sécurité garantissant qu'un nouveau noeud ne pourra pas déchiffrer avec la nouvelle clé de chiffrement les anciens messages chiffrés avec une clé de chiffrement précédente.

Définition 12 (Forward Secrecy)**.** La *Forward Secrecy* est une propriété de sécurité garantissant qu'un nouveau noeud ne pourra pas déchiffrer avec la nouvelle clé de chiffrement les futurs messages chiffrés avec une prochaine clé de chiffrement.

Une copie locale du document est sauvegardée dans le navigateur, avec l'ensemble des modifications. L'utilisateur-rice peut ainsi accéder à ses documents même sans connexion internet, pour les consulter ou modifier. Les modifications effectuées dans ce mode hors-ligne seront partagées aux collaborateur-rices à la prochaine connexion de l'utilisateur-rice.

Finalement, la page d'accueil de l'application permet aussi de lister ses documents. L'utilisateur-rice peut ainsi facilement parcourir ses documents, récupérer leur url pour les partager ou encore supprimer leur copie locale. Figure 2.2 illustre cette page de l'application.

2.1.3 Architecture système

Nous représentons l'architecture système d'une collaboration utilisant MUTE par la Figure 2.3.

Plusieurs types de noeuds composent cette architecture. Nous décrivons ci-dessous le type de chacun de ces noeuds ainsi que leurs rôles.

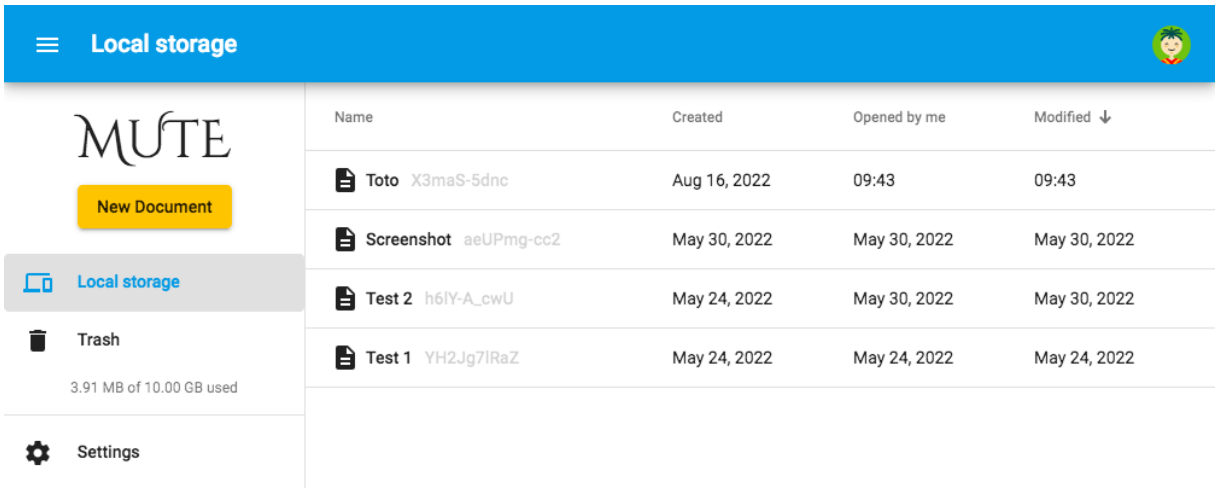


FIGURE 2.2 – Capture d'écran de la liste des documents.

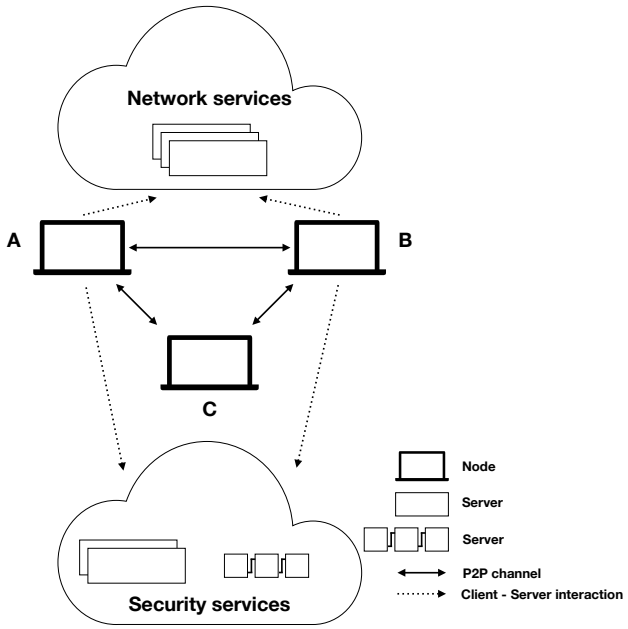


FIGURE 2.3 – Architecture système de l'application MUTE

Pairs

Au centre de la collaboration se trouvent les noeuds qui correspondent aux utilisateurrices de l'application et à leurs appareils. Chaque noeud correspond à une instance de l'application MUTE, c.-à-d. l'éditeur collaboratif de texte. Chacun de ces noeuds peut donc consulter des documents et les modifier.

Ces noeuds forment un réseau P2P, qui leur permet d'échanger directement notamment pour diffuser les modifications effectuées sur le document. Les pairs interagissent aussi avec les autres types de noeuds, que nous décrivons dans les parties suivantes.

Notons qu'un noeud peut toutefois être déconnecté du système, c.-à-d. dans l'incapa-

cité de se connecter aux autres pairs et d'interagir avec les autres types de noeuds. Cela ne l'empêche toutefois pas l'utilisateur-rice d'utiliser MUTE.

Services réseau

Nous décrivons par cette appelation l'ensemble des composants nécessaires à l'établissement et le bon fonctionnement du réseau P2P entre les appareils des utilisateur-rices.

Il s'agit de serveurs ayant pour buts de :

(i) Permettre à un pair d'obtenir les informations sur son propre état nécessaires pour l'établissement de connexions P2P.

(ii) Permettre à un pair de découvrir les autres pairs travaillant sur le même document et d'établir une connexion avec eux.

(iii) Permettre à des pairs de communiquer même si leur configurations réseaux respectives empêchent l'établissement d'une connection P2P directe.

Nous détaillons plus précisément chacun de ces services et les interactions entre les pairs et ces derniers dans la section 2.4.

Services sécurité

Nous décrivons par cette appelation l'ensemble des composants nécessaires à l'authentification des utilisateur-rices et à l'établissement de clés de groupe de chiffrement.

Il s'agit de serveurs ayant pour buts de :

(i) Permettre à un pair de s'authentifier.

(ii) Permettre à un pair de faire connaître sa clé publique de chiffrement.

(iii) Vérifier l'identité d'un pair.

(iv) Permetttre à un pair de vérifier le comportement honnête du ou des serveurs servant les clés publiques de chiffrement.

Nous dédions la section 2.5 à la description de ces différents services et les interactions des pairs avec ces derniers.

2.1.4 Architecture logicielle

Nous décrivons l'architecture logicielle d'un pair, c.-à-d. d'une instance de l'application MUTE dans un navigateur, dans la Figure 2.4.

Cette architecture logicielle se compose de plusieurs composants, que nous regroupons par couche. Chacune de ces couches possède un rôle, que nous présentons brièvement ci-dessous avant de les décrire de manière plus détaillée dans leur section respective.

(i) La couche *interface utilisateur*, qui regroupe l'ensemble des composants permettant de communiquer des informations aux pairs et avec lesquelles ils peuvent interagir, c.-à-d. le document lui-même, son titre mais aussi la liste des collaborateur-rices actuellement connectés. Cette couche se charge de transmettre les actions de l'utilisateur-rice aux couches inférieures, et inversement de présenter à l'utilisateur-rice les modifications effectuées par ses pairs.

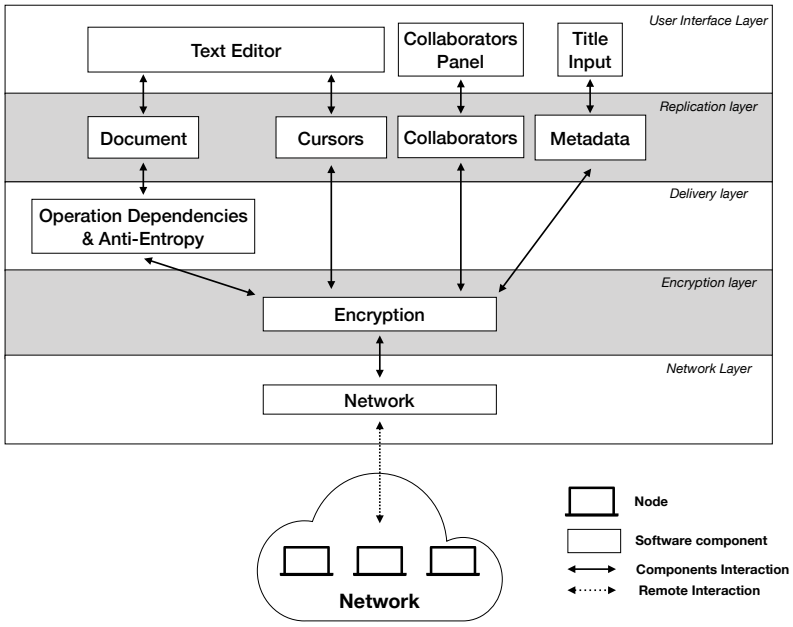


FIGURE 2.4 – Architecture logicielle de l'application MUTE

(ii) La couche *réplication*, qui regroupe l'ensemble des composants permettant de représenter les données répliquées entre pairs, c.-à-d. les CRDTs utilisés pour représenter le document, ses métadonnées (titre, date de création...), l'ensemble des collaborateurs et leur curseur. Cette couche se charge d'intégrer les modifications effectuées par l'utilisateur-rice et de transmettre les opérations correspondantes aux couches inférieures, et inversement d'intégrer les opérations effectuées par ses pairs et d'indiquer à la couche *interface utilisateur* les modifications correspondantes.

(iii) La couche *livraison*, qui est constitué d'un unique composant permettant de garantir les modèles de livraison requis par les différents CRDTs implémentés pour représenter le document. Cette couche se charge d'adjoindre aux opérations de l'utilisateur-rice leur(s) dépendance(s) avant de les transmettre aux couches inférieures, et de livrer les opérations de ses pairs une fois leur(s) dépendance(s) livrées au préalable, ou de les mettre en attente le cas échéant.

(iv) La couche *sécurité*, qui est constitué d'un unique composant gérant le chiffrement des messages. Cette couche se charge d'établir la clé de chiffrement de groupe, puis de chiffrer les messages de l'utilisateur-rice avec cette dernière avant de les transmettre à la couche inférieure, et inversement de déchiffrer les messages chiffrés de ses pairs avant de les transmettre aux couches supérieures.

(v) La couche *réseau*, qui est constitué d'un unique composant permettant d'interagir avec le réseau P2P. Cette couche se charge d'établir les connexions P2P, puis permet de diffuser les messages chiffrés de l'utilisateur-rice à un ou plusieurs de ses pairs, et inversement de transmettre les messages chiffrés de ses pairs à la couche supérieure.

2.2 Couche interface utilisateur

Comme illustré par la Figure 2.1, l'interface de la page d'un document se compose d'un éditeur de texte supportant le langage de balisage Markdown. Ainsi, l'éditeur permet d'inclure plusieurs éléments légers de style. Les balises du langage Markdown étant du texte, elles sont répliquées nativement par la structure de données utilisée en interne par MUTE.

L'éditeur est agrémenté de plusieurs mécanismes permettant d'établir une conscience de groupe entre les collaborateur-rices. L'indicateur en haut à droite de la fenêtre représente le statut de connexion de l'utilisateur. Ceci lui indique s'il est actuellement connecté au réseau P2P, en cours de connexion, ou si un incident réseau a lieu.

MUTE affiche la liste des collaborateur-rices actuellement connectés sur la droite de l'éditeur. De plus, un curseur ou une sélection distante est associée à chaque collaborateur de la liste. Elle permet d'indiquer à l'utilisateur courant dans quelles sections du document ses collaborateur-rices sont en train de travailler. Ainsi, ils peuvent se répartir la rédaction du document de manière implicite ou suivre facilement les modifications d'un collaborateur-rice.

Les documents de l'utilisateur étant sauvegardés dans le navigateur, les documents sont aussi bien disponibles en étant en ligne que hors ligne. Une seconde page, listant les documents sauvegardés, permet à l'utilisateur de parcourir ses différents documents.

2.3 Couche réplication

2.3.1 Modèle de données du document texte

MUTE propose plusieurs alternatives pour représenter le document texte. MUTE permet de soit utiliser une implémentation de LogootSplit [6], soit de RenamableLogootSplit[6] ou soit de Dotted LogootSplit [7]. Ce choix est effectué via une valeur de configuration de l'application choisie au moment de son déploiement.

Le modèle de données utilisé interagit avec l'éditeur de texte par l'intermédiaire d'opérations textes. Lorsque l'utilisateur effectue des modifications locales, celles-ci sont détectées et mises sous la forme d'opérations textes. Elles sont transmises au modèle de données, qui les intègre alors à la structure de données répliquées. Le CRDT retourne en résultat l'opération distante à propager aux autres noeuds.

De manière complémentaire, lorsqu'une opération distante est livrée au modèle de données, elle est intégrée par le CRDT pour actualiser son état. Le CRDT génère les opérations textes correspondantes et les transmet à l'éditeur de texte pour mettre à jour la vue.

En plus du texte, MUTE maintient un ensemble de métadonnées par document. Par exemple, les utilisateurs peuvent donner un titre au document. Pour représenter cette

6. Les deux implémentations proviennent de la librairie `mute-structs` : `https://github.com/coast-team/mute-structs`

7. Implémentation fournie par la librairie suivante : `https://github.com/coast-team/dotted-logootsplit`

donnée additionnelle, nous associons un Last-Writer-Wins Register CRDT synchronisé par opérations [22] au document. De façon similaire, nous utilisons un First-Writer-Wins Register CRDT synchronisé par opérations pour représenter la date de création du document.

2.3.2 Module de livraison des opérations

Dans le cadre de LogootSplit et de RenamableLogootSplit, le modèle de données utilisé pour représenter le document texte est couplé au composant `Sync`. Le rôle de ce composant est d'assurer le respect du modèle de livraison des opérations au CRDT. Pour cela, le module `Sync` doit implémenter les contraintes présentées dans la **??** et dans la **??**.

Livraison des opérations en exactement un exemplaire

Afin de respecter la contrainte de *exactly-once delivery*, il est nécessaire d'identifier de manière unique chaque opération. Pour cela, le module `Sync` ajoute un *Dot* [51] à chaque opération :

Définition 13 (Dot)**.** Un *Dot* est une paire ⟨nodeId, nodeSyncSeq⟩ où

- nodeId est l'identifiant unique du noeud qui a généré l'opération.
- nodeSyncSeq est le numéro de séquence courant du noeud à la génération de l'opération.

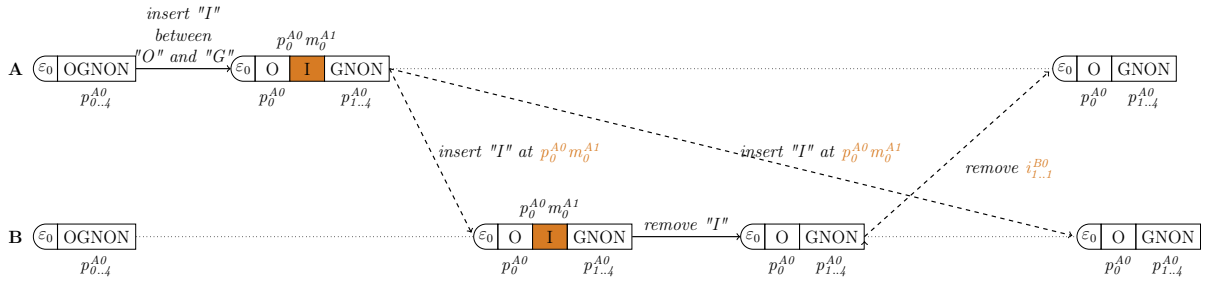
Il est à noter que *nodeSyncSeq* est différent du *nodeSeq* utilisé dans LogootSplit et RenamableLogootSplit (cf. **??**, page **??**). En effet, *nodeSyncSeq* se doit d'augmenter à chaque opération tandis que *nodeSeq* n'augmente qu'à la création d'un nouveau bloc. Les contraintes étant différentes, il est nécessaire de distinguer ces deux données.

Chaque noeud maintient une structure de données représentant l'ensemble des opérations reçues par le pair. Elle permet de vérifier à la réception d'une opération si le dot de cette dernière est déjà connu. S'il s'agit d'un nouveau dot, le module `Sync` peut livrer l'opération au CRDT et ajouter son dot à la structure. Le cas échéant, cela indique que l'opération a déjà été livrée précédemment et doit être ignorée cette fois-ci.

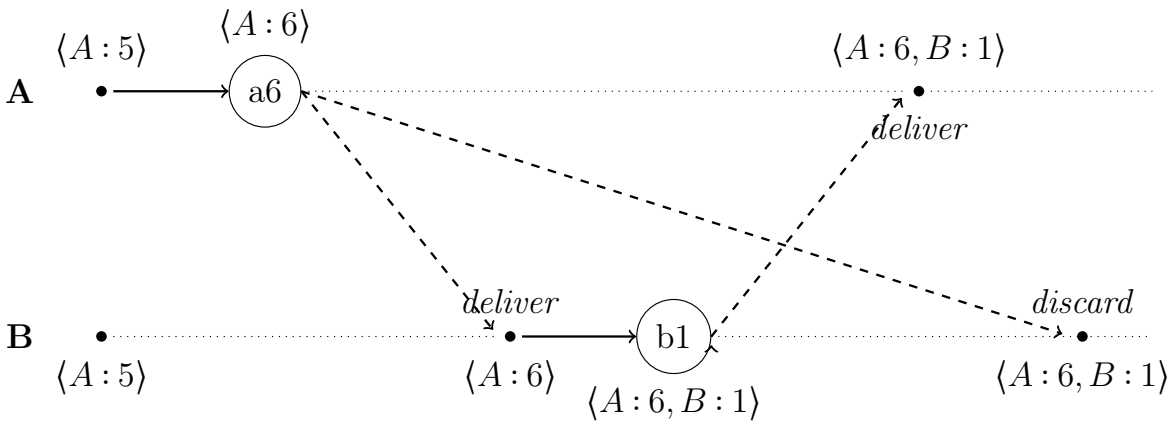
Plusieurs structures de données sont adaptées pour maintenir l'ensemble des opérations reçues. Dans le cadre de MUTE, nous avons choisi d'utiliser un vecteur de versions. Cette structure nous permet de réduire à un dot par noeud le surcoût en métadonnées du module `Sync`, puisqu'il ne nécessite que de stocker le dot le plus récent par noeud. Cette structure permet aussi de vérifier en temps constant si une opération est déjà connue. La Figure 2.5 illustre son fonctionnement.

Dans cet exemple, qui reprend celui de la **??**, deux noeuds A et B répliquent une séquence. Initialement, celle-ci contient les éléments "OGNON". Ces éléments ont été insérés un par un par le noeud A, donc par le biais des opérations *a1* à *a5*. Le module `Sync` de chaque noeud maintient donc initialement le vecteur de version $\langle A : 5 \rangle$.

Le noeud A insère l'élément "I" entre les éléments "O" et "G". Cette modification est alors labellisée *a6* par son module `Sync` et est envoyée au noeud B. À la réception de cette opération, le module `Sync` de B compare son dot avec son vecteur de version local.



(a) Exécution avec livraison multiple d'une opération *insert*



(b) État et comportement de la couche `Sync` au cours de l'exécution décrite en Figure 2.5a

FIGURE 2.5 – Gestion de la livraison *exactly-once* des opérations

L'opération *a6* étant la prochaine opération attendue de A, celle-ci est acceptée : elle est alors livrée au CRDT et le vecteur de version est mis à jour.

Le noeud B supprime ensuite l'élément nouvellement inséré. S'agissant de la première modification de B, cette modification *b1* ajoute l'entrée correspondante dans le vecteur de version $\langle A:6, B:1\rangle$. L'opération est envoyée au noeud A. Cette opération étant la prochaine opération attendue de B, elle est acceptée et livrée.

Finalement, le noeud B reçoit de nouveau l'opération *a6*. Son module `Sync` détermine alors qu'il s'agit d'un doublon : l'opération apparaît déjà dans le vecteur de version $\langle A:6, B:1\rangle$. L'opération est donc ignorée, et la résurgence de l'élément "I" illustrée dans la **??** est évitée.

Il est à noter que dans le cas où un noeud reçoit une opération avec un dot plus élevé que celui attendu (e.g. le noeud A reçoit une opération *b3* à la fin de l'exemple), cette opération est mise en attente. En effet, livrer cette opération nécessiterait de mettre à jour le vecteur de version à $\langle A:6, B:3\rangle$ et masquerait le fait que l'opération *b2* n'a jamais été reçue. L'opération *b3* serait donc mise en attente jusqu'à la livraison de l'opération *b2*.

Ainsi, l'implémentation de livraison *exactly-once* avec un vecteur de version comme structure de données force une livraison First In, First Out (FIFO) des opérations par noeuds. Il s'agit d'une contrainte non-nécessaire et qui peut introduire des délais dans la collaboration, notamment si une opération d'un noeud est perdue par le réseau. Nous

jugeons cependant acceptable ce compromis entre le surcoût du mécanisme de livraison *exactly-once* et son impact sur l'expérience utilisateur.

Pour retirer cette contrainte superflue, il est possible de remplacer cette structure de données par un *Interval Version Vector* [52]. Au lieu d'enregistrer seulement le dernier dot intégré par noeud, cette structure de données enregistre les intervalles de dots intégrés. Ceci permet une livraison *out of order* des opérations tout en garantissant une livraison *exactly-once* et en compactant efficacement les données stockées par le module `Sync` à terme.

Livraison de l'opération *remove* après l'opération *insert*

La seconde contrainte que le modèle de livraison doit respecter spécifie qu'une opération *remove* doit être livrée après les opérations *insert* insérant les éléments concernés.

Pour cela, le module `Sync` ajoute un ensemble *Deps* à chaque opération *remove* avant de la diffuser :

Définition 14 (Deps)**.** *Deps* est un ensemble d'opérations. Il représente l'ensemble des opérations dont dépend l'opération *remove* et qui doivent donc être livrées au préalable.

Plusieurs structures de données sont adaptées pour représenter les dépendances de l'opération *remove*. Dans le cadre de MUTE, nous avons choisi d'utiliser un ensemble de dots : pour chaque élément supprimé par l'opération *remove*, nous identifions le noeud l'ayant inséré et nous ajoutons le dot correspondant à l'opération la plus récente de ce noeud à l'ensemble des dépendances. Cette approche nous permet de limiter à un dot par élément supprimé le surcoût en métadonnées des dépendances et de les calculer en un temps linéaire par rapport au nombre d'éléments supprimés. Nous illustrons le calcul et l'utilisation des dépendances de l'opération *remove* à l'aide de la Figure 2.6.

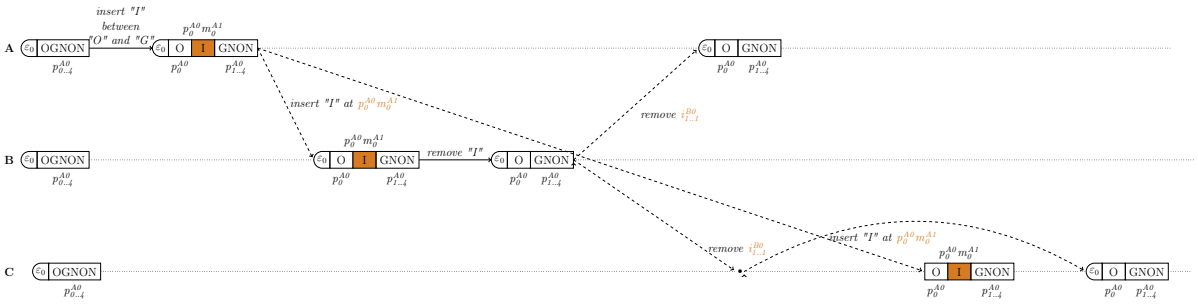
Cet exemple reprend et complète celui de la Figure 2.6. Trois noeuds A, B et C répliquent et éditent collaborativement une séquence. Les trois noeuds partagent le même état initial : une séquence contenant les éléments "OGNON" et un vecteur de version $\langle A : 5 \rangle$.

Le noeud A insère l'élément "I" entre les éléments "O" et "G". Cet élément se voit attribué l'identifiant $p_0^{A0} m_0^{A1}$. L'opération correspondante *a6* est diffusée aux autres noeuds.

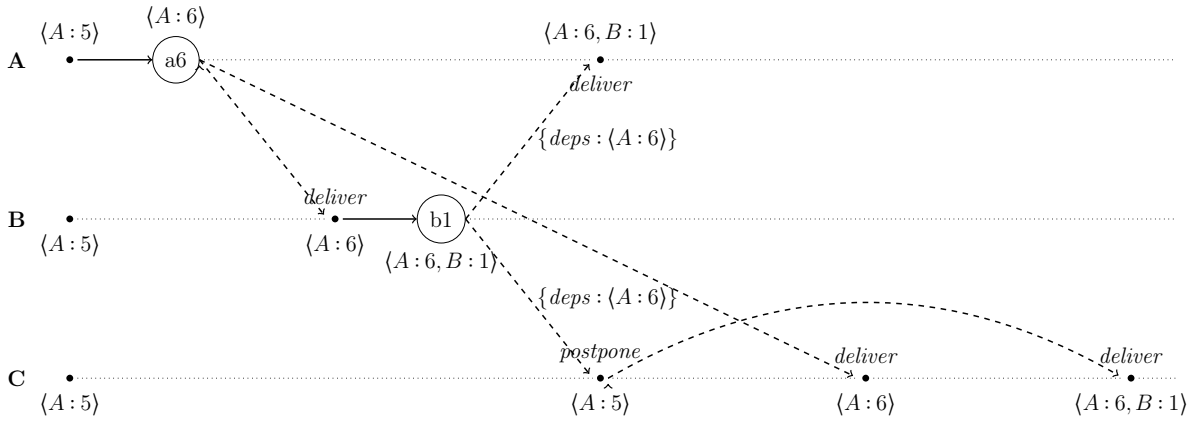
À la réception de cette dernière, le noeud B supprime l'élément "I" nouvellement inséré et génère l'opération *b1* correspondante. Comme indiqué précédemment, l'opération *b1* étant une opération *remove*, le module `Sync` calcule ses dépendances avant de la diffuser. Pour chaque élément supprimé ("I"), `Sync` récupère l'identifiant de l'élément ($p_0^{A0} m_0^{A1}$) et en extrait l'identifiant du noeud qui l'a inséré (A). `Sync` ajoute alors le dot de l'opération la plus récente reçue de ce noeud ($\langle A : 6 \rangle$) à l'ensemble des dépendances de l'opération. L'opération est ensuite diffusée.

À la réception de l'opération *b1*, le noeud A vérifie s'il possède l'ensemble des dépendances de l'opération. Le noeud A ayant déjà intégré l'opération *a6*, le module `Sync` livre l'opération *b1* au CRDT.

À l'inverse, lorsque le noeud C reçoit l'opération *b1*, il n'a pas encore reçu l'opération *a6*. L'opération *b1* est alors mise en attente. À la réception de l'opération *a6*, celle-ci est



(a) Exécution avec livraison dans le désordre d'une insertion et de sa suppression



(b) État et comportement de la couche Sync au cours de l'exécution décrite en Figure 2.6a

FIGURE 2.6 – Gestion de la livraison *causale-remove* des opérations

livrée. Le module Sync ré-évalue alors le cas de l'opération *b1* et détermine qu'elle peut à présent être livrée.

Il est à noter que notre approche pour générer l'ensemble des dépendances est une approximation. En effet, nous ajoutons les dots des opérations les plus récentes des auteurs des éléments supprimés. Nous n'ajoutons pas les dots des opérations qui ont spécifiquement insérés les éléments supprimés. Pour cela, il serait nécessaire de parcourir le log des opérations à la recherche des opérations *insert* correspondante. Cette méthode serait plus coûteuse, sa complexité dépendant du nombre d'opérations dans le log d'opérations, et incompatible avec un mécanisme tronquant le log des opérations en utilisant la stabilité causale. Notre approche introduit un potentiel délai dans la livraison d'une opération *remove* par rapport à une livraison utilisant ses dépendances exactes, puisqu'elle va reposer sur des opérations plus récentes et potentiellement encore inconnues par le noeud. Mais il s'agit là aussi d'un compromis que nous jugeons acceptable entre le surcoût du mécanisme de livraison et l'expérience utilisateur.

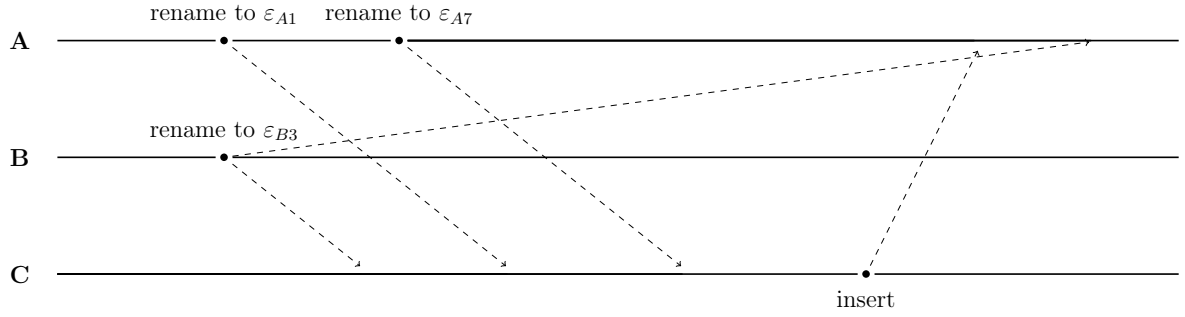
Livraison des opérations après l'opération *rename* introduisant leur époque

La troisième contrainte spécifiée par le modèle de livraison est qu'une opération doit être livrée après l'opération *rename* qui a introduite son époque de génération.

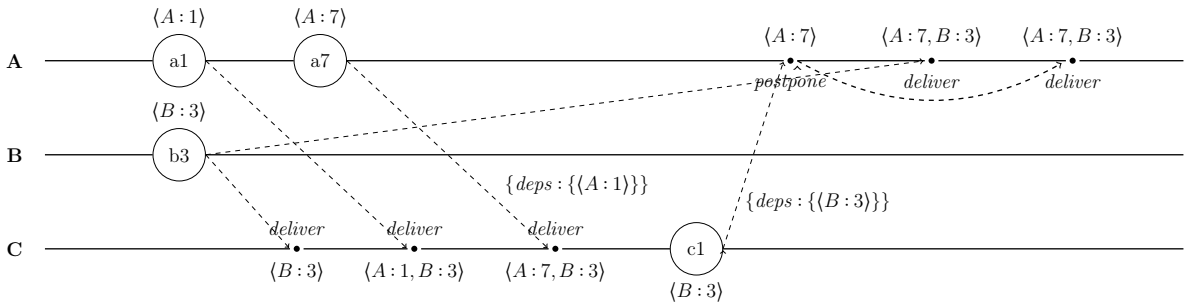
Pour cela, le module Sync doit donc récupérer l'époque courante de la séquence répli-

quée, récupérer le dot de l'opération *rename* l'ayant introduite et l'ajouter en tant que dépendance de chaque opération. Cependant, dans notre implémentation, le module `Sync` et le module représentant la séquence répliquée sont découplés et ne peuvent interagir directement l'un avec l'autre.

Pour remédier à ce problème, le module `Sync` maintient une structure supplémentaire : un vecteur des dots des opérations *rename* connues. À la réception d'une opération *rename* distante, l'entrée correspondante de son auteur est mise à jour avec le dot de la nouvelle époque introduite. À la génération d'une opération locale, l'opération est examinée pour récupérer son époque de génération. `Sync` conserve alors seulement l'entrée correspondante dans le vecteur des dots des opérations *rename*. À ce stade, le contenu du vecteur est ajouté en tant que dépendance de l'opération. Ensuite, si l'opération locale s'avère être une opération *rename*, le vecteur est modifié pour ne conserver que le dot de l'époque introduite par l'opération. La Figure 2.7 illustre ce fonctionnement.



(a) Exécution



(b) État et comportement de la couche Sync au cours de l'exécution décrite en Figure 2.7a

FIGURE 2.7 – Gestion de la livraison *epoch based* des opérations

Dans la Figure 2.7a, nous décrivons une exécution suivante en ne faisant apparaître que les opérations importantes : les opérations *rename* et une opération *insert* finale. Dans cette exécution, trois noeuds A, B et C répliquent et éditent collaborativement une séquence. Initialement, aucune opération *rename* n'a encore eu lieu. Le noeud A effectue une première opération *rename* (*a1*) puis une seconde opération *rename* (*a7*), et les diffuse. En concurrence, le noeud B génère et propage sa propre opération *rename* (*b3*). De son côté, le noeud C reçoit les opérations *b3*, puis *a1* et *a7*. Il émet ensuite une opération *insert* (*c1*). Le noeud A reçoit cette opération avant de finalement recevoir l'opération *b3*.

Dans la Figure 2.7b, nous faisons apparaître l'état de `Sync` et les décisions prises par ce dernier au cours de l'exécution. Initialement, le vecteur des dots des opérations *rename* connues est vide. Ainsi, lorsque A génère l'opération *a1*, celle-ci ne se voit ajouter aucune dépendance (nous ne représentons pas les dépendances des opérations qui correspondent à l'ensemble vide). A met ensuite à jour son vecteur des dots des opérations *rename* avec le dot $\langle A : 1 \rangle$. B procède de manière similaire avec l'opération *b3*.

Quand A génère l'opération *a7*, le dot $\langle A : 1 \rangle$ est ajouté en tant que dépendance. Le dot $\langle A : 7 \rangle$ remplace ensuite ce dernier dans le vecteur des dots des opérations *rename*.

À la réception de l'opération *b3*, le module `Sync` de C peut la livrer au CRDT, l'ensemble de ses dépendances étant vérifié. Le noeud C ajoute alors à son vecteur des dots des opérations *rename* le dot $\langle B : 3 \rangle$. Il procède de même pour l'opération *a1* : il la livre et ajoute le dot $\langle A : 1 \rangle$. Le module `Sync` ne connaissant pas l'époque courante de la séquence répliquée, il maintient les deux dots localement.

Lorsque le noeud C reçoit l'opération *a7*, l'ensemble de ses contraintes est vérifié : l'opération *a1* a été livrée précédemment. L'opération est donc livrée et le vecteur de dots des opérations *rename* mis à jour avec $\langle A : 7 \rangle$.

Quand le noeud C effectue l'opération locale *c1*, le module `Sync` obtient l'information de l'époque courante de la séquence : ε_{b3}. C met à jour son vecteur de dots des opérations *rename* pour ne conserver que l'entrée du noeud B : $\langle B : 3 \rangle$. Ce dot est ajouté en tant que dépendance de l'opération *c1* avant sa diffusion.

À la réception de l'opération *c1* par le noeud A, cette opération est mise en attente par `Sync`, l'opération *b3* n'ayant pas encore été livrée. Le noeud reçoit ensuite l'opération *b3*. Son vecteur des dots des opérations *rename* est mis à jour et l'opération livrée. Les conditions pour l'opération *c1* étant désormais remplies, l'opération est alors livrée.

Cette implémentation de la contrainte de la livraison *epoch-based* dispose de plusieurs avantages : sa complexité spatiale dépend linéairement du nombre de noeuds et les opérations de mise à jour du vecteur des dots des opérations *rename* s'effectuent en temps constant. De plus, seul un dot est ajouté en tant que dépendance des opérations, la taille du vecteur des dots étant ramené à 1 au préalable. Finalement, cette implémentation ne contraint pas une livraison causale des opérations *rename* et permet donc de les appliquer dès que possible.

Livraison des opérations à terme

La dernière contrainte du modèle de livraison précise que toutes les opérations doivent être livrées à tous les noeuds à terme. Cependant, le réseau étant non-fiable, des messages peuvent être perdus au cours de l'exécution. Il est donc nécessaire que les noeuds rediffusent les messages perdus pour assurer leur livraison à terme.

Pour cela, nous implémentons un mécanisme d'anti-entropie basé sur [53]. Ce mécanisme permet à un noeud source de se synchroniser avec un autre noeud cible. Il est executé par l'ensemble des noeuds de manière indépendante. Nous décrivons ci-dessous son fonctionnement.

De manière périodique, le noeud choisit un autre noeud cible de manière aléatoire. Le noeud source lui envoie alors une représentation de son état courant, c.-à-d. son vecteur de version.

À la réception de ce message, le noeud cible compare le vecteur de version reçu par rapport à son propre vecteur de version. À partir de ces données, il identifie les dots des opérations de sa connaissance qui sont inconnues au noeud source. Grâce à leur dot, le noeud cible retrouve ces opérations depuis son log des opérations. Il envoie alors une réponse composée de ces opérations au noeud source.

À la réception de la réponse, le noeud source intègre normalement les opérations reçues. La Figure 2.8 illustre ce mécanisme.

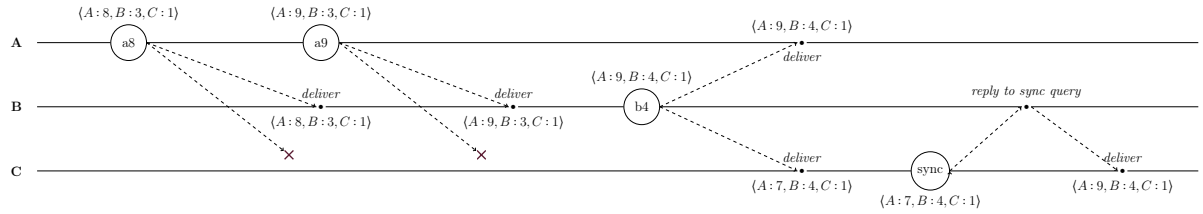


FIGURE 2.8 – Utilisation du mécanisme d'anti-entropie par le noeud C pour se synchroniser avec le noeud B

Dans cette figure, nous représentons une exécution à laquelle participent trois noeuds : A, B et C. Initialement, les trois noeuds sont synchronisés. Leur vecteurs de version sont identiques et ont pour valeur $\langle A:7, B:3, C:1\rangle$.

Le noeud A effectue les opérations *a8* puis *a9* et les diffusent sur le réseau. Le noeud B reçoit ces opérations et les livre à son CRDT. Il effectue ensuite et propage l'opération *b4*, qui est reçue et livrée par A. Ils atteignent tous deux la version représenté par le vecteur $\langle A:9, B:4, C:1\rangle$

De son côté, le noeud C ne reçoit pas les opérations *a8* et *a9* à cause d'une défaillance réseau. Néanmoins, cela ne l'empêche pas de livrer l'opération *b4* à sa réception et d'obtenir la version $\langle A:7, B:4, C:1\rangle$.

Le noeud C déclenche ensuite son mécanisme d'anti-entropie. Il choisit aléatoirement le noeud B comme noeud cible. Il lui envoie un message de synchronisation avec pour contenu le vecteur de version $\langle A:7, B:8, C:1\rangle$.

À la réception de ce message, le noeud B compare ce vecteur avec le sien. Il détermine que le noeud C n'a pas reçu les opérations *a8* et *a9*. B les récupère depuis son log des opérations et les envoie à C par le biais d'un nouveau message.

À la réception de la réponse de B, le noeud C livre les opérations *a8* et *a9*. Il atteint alors le même état que A et B, représenté par le vecteur de version $\langle A:9, B:4, C:1\rangle$.

Ce mécanisme d'anti-entropie nous permet ainsi de garantir la livraison à terme de toutes les opérations et de compenser les défaillances du réseau. Il nous sert aussi de mécanisme de synchronisation : à la connexion d'un pair, celui-ci utilise ce mécanisme pour récupérer les opérations effectuées depuis sa dernière connexion. Dans le cas où il s'agit de la première connexion du pair, il lui suffit d'envoyer un vecteur de version vide pour récupérer l'intégralité des opérations.

Ce mécanisme propose plusieurs avantages. Son exécution n'implique que le noeud source et le noeud cible, ce qui limite les coûts de coordination. De plus, si une défaillance a lieu lors de l'exécution du mécanisme (perte d'un des messages, panne du noeud cible...),

cette défaillance n'est pas critique : le noeud source se synchronisera à la prochaine exécution du mécanisme. Ensuite, ce mécanisme réutilise le vecteur de version déjà nécessaire pour la livraison *exactly-once*, comme présenté en section 2.3.2. Il ne nécessite donc pas de stocker une nouvelle structure de données pour détecter les différences entre noeuds.

En contrepartie, la principale limite de ce mécanisme d'anti-entropie est qu'il nécessite de maintenir et de parcourir périodiquement le log des opérations pour répondre aux requêtes de synchronisation. La complexité spatiale et en temps du mécanisme dépend donc linéairement du nombre d'opérations. Qui plus est, nous sommes dans l'incapacité de tronquer le log des opérations en se basant sur la stabilité causale des opérations puisque nous utilisons ce mécanisme pour mettre à niveau les nouveaux pairs. À moins de mettre en place un mécanisme de compression du log comme évoqué en **??**, ce log des opérations croit de manière monotone. Néanmoins, une alternative possible est de mettre en place un système de chargement différé des opérations pour ne pas surcharger la mémoire.

2.3.3 Collaborateur-rices

Pour assurer la qualité de la collaboration même à distance, il est important d'offrir des fonctionnalités de conscience de groupe aux utilisateurs. Une de ces fonctionnalités est de fournir la liste des collaborateur-rices actuellement connectés. Les protocoles d'appartenance au réseau sont une catégorie de protocoles spécifiquement dédiée à cet effet. Ainsi, nous devions en implémenter un dans MUTE.

MUTE présente cependant plusieurs contraintes liées à notre modèle du système que le protocole sélectionné doit respecter. Tout d'abord, le protocole doit être compatible avec un environnement P2P, où les noeuds partagent les mêmes droits et responsabilités. De plus, le protocole doit présenter une capacité de passage à l'échelle pour être adapté aux collaborations à large échelle.

En raison de ces contraintes, notre choix s'est porté sur le protocole SWIM [33]. Proposé par Das et al., ce protocole d'appartenance au réseau offre les propriétés intéressantes suivantes. Tout d'abord, le nombre de messages diffusés sur le réseau est proportionnel de façon linéaire au nombre de pairs. Pour être plus précis, le nombre de messages envoyés par un pair par période du protocole est constant. De plus, il fournit à chaque noeud une vue de la liste des collaborateur-rices cohérente à terme, même en cas de réception désordonnée des messages du protocoles. Finalement, il intègre un mécanisme permettant de réduire le taux de faux positifs, c.-à-d. le taux de pairs déclarés injustement comme défaillants.

Pour cela, SWIM découple les deux composants d'un protocole d'appartenance au réseau : le mécanisme de *détection des défaillances des pairs* et le mécanisme de *dissémination des mises à jour du groupe*.

Mécanisme de détection des défaillances des pairs

Le mécanisme de détection des défaillances des pairs est executé de manière périodique, toutes les T unités de temps, par chacun des noeuds du système de manière non-coordonnée. Son fonctionnement est illustré par la Figure 2.9.

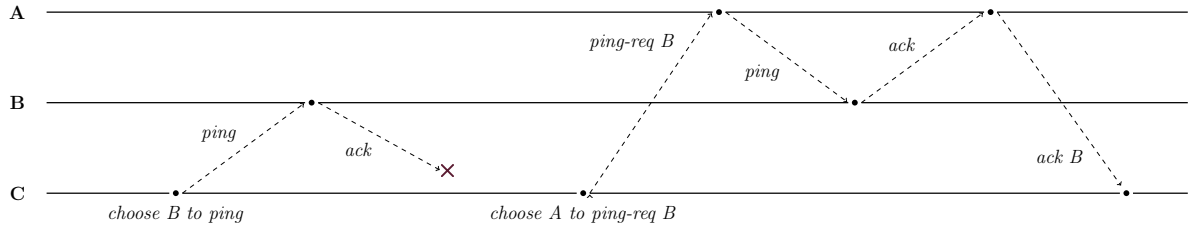


FIGURE 2.9 – Exécution du mécanisme de détection des défaillances par le noeud C pour tester le noeud B

Dans cet exemple, le réseau est composé des trois noeuds A, B et C. Le noeud C démarre l'exécution du mécanisme de détection des défaillances.

Tout d'abord, le noeud C sélectionne un noeud cible de manière aléatoire, ici B, et lui envoie un message *ping*. À la réception de ce message, le noeud B lui signifie qu'il est toujours opérationnel en lui répondant avec un message *ack*. À la réception de ce message par C, cette exécution du mécanisme de détection des défaillances prendrait fin. Mais dans l'exemple présenté ici, ce message est perdu par le réseau.

En l'absence de réponse de la part de B au bout d'un temps spécifié au préalable, le noeud C passe à l'étape suivante du mécanisme. Le noeud C sélectionne un autre noeud, ici A, et lui demande de vérifier via le message *ping-req B* si B a eu une défaillance. À la réception de la requête de ping, le noeud A envoie un message *ping* à B. Comme précédemment, B répond au *ping* par le biais d'un *ack* à A. A informe alors C du bon fonctionnement du B via le message *ack B*. Le mécanisme prend alors fin, jusqu'à sa prochaine exécution.

Si C n'avait pas reçu de réponse suite à sa *ping-req B* envoyée à A, C aurait supposé que B a eu une défaillance. Afin de réduire le taux de faux positifs, SWIM ne considère pas directement les noeuds n'ayant pas répondu comme en panne : ils sont tout d'abord *suspectés* d'être en panne. Après un certain temps sans signe de vie d'un noeud suspecté d'être en panne, le noeud est *confirmé* comme défaillant.

L'information qu'un noeud est suspecté d'être en panne est propagé dans le réseau via le mécanisme de dissémination des mises à jour du groupe décrit ci-dessous. Si un noeud apprend qu'il est suspecté d'une panne, il dissémine à son tour l'information qu'il est toujours opérationnel pour éviter d'être confirmé comme défaillant.

Pour éviter qu'un message antérieur n'invalide une suspicion d'une défaillance et retarde ainsi sa détection, SWIM introduit un numéro d'*incarnation*. Chaque noeud maintient un numéro d'incarnation. Lorsqu'un noeud apprend qu'il est suspecté d'une panne, il incrémente son numéro d'incarnation avant de propager l'information contradictoire.

Ainsi, afin de représenter la liste des collaborateur-rices, le protocole SWIM utilise la structure de données présentée par la Définition 15 :

Définition 15 (Liste des collaborateur-rices)**.** La *liste des collaborateur-rices* est un ensemble de triplets $\langle nodeId, nodeStatus, nodeIncarn \rangle$ où

— *nodeId* correspond à l'identifiant du noeud correspondant à ce tuple.

— *nodeStatus* correspond au statut courant du noeud correspondant à ce tuple, c.-à-d. *Alive* s'il est considéré comme opérationnel, *Suspect* s'il est suspecté d'une

défaillance, *Confirm* s'il est considéré comme défaillant.

— *nodeIncarn* correspond au numéro d'incarnation maximal, c.-à-d. le plus récent, connu pour le noeud correspondant à ce tuple.

Chaque noeud réplique cette liste et la fait évoluer au cours de l'exécution du mécanisme présenté jusqu'ici. Lorsqu'une mise à jour est effectuée, celle-ci est diffusée de la manière présentée ci-dessous.

Mécanisme de dissémination des mises à jour du groupe

Quand l'exécution du mécanisme de détection des défaillances par un noeud met en lumière une évolution de la liste des collaborateur-rices, cette mise à jour doit être propagée au reste des noeuds.

Or, diffuser cette mise à jour à l'ensemble du réseau serait coûteux pour un seul noeud. Afin de propager cette information de manière efficace, SWIM propose d'utiliser un protocole de diffusion épidémique : le noeud transmet la mise à jour qu'à un nombre réduit λ [8] de pairs, qui se chargeront de la transmettre à leur tour. Le mécanisme de dissémination des mises à jour de SWIM fonctionne donc de la manière suivante.

Chaque mise à jour du groupe est stockée dans une liste et se voit attribuer un compteur, initialisé avec $\lambda \log n$ avec n le nombre de noeuds. À chaque génération d'un message pour le mécanisme de détection des défaillances, un nombre arbitraire de mises à jour sont sélectionnées dans la liste et attachées au message. Leur compteurs respectifs sont décrémentés. Une fois que le compteur d'une mise à jour atteint 0, celle-ci est retirée de la liste.

À la réception d'un message, le noeud le traite comme définit précédemment en section 2.3.3. De manière additionnelle, il intègre dans sa liste des collaborateur-rices les mises à jour attachées au message en utilisant la règle suivante :

$$\forall i, j, k \cdot i \leq j \cdot \langle Alive, i\rangle < \langle Suspect, j\rangle < \langle Confirm, k\rangle$$

Ainsi, le mécanisme de dissémination des mises à jour du groupe réutilise les messages du mécanisme de détection des défaillances pour diffuser les modifications. Cela permet de propager les évolutions de la liste des collaborateur-rices sans ajouter de message supplémentaire. De plus, les règles de précédence sur l'état d'un collaborateur permettent aux noeuds de converger même si les mises à jour sont reçues dans un ordre distinct.

Modifications apportées

Nous avons ensuite apporté plusieurs modifications à la version du protocole SWIM présentée dans [33]. Notre première modification porte sur l'ordre de priorité entre les états d'un pair.

8. [33] montre que choisir une valeur constante faible comme λ suffit néanmoins à garantir la dissémination des mises à jour à l'ensemble du réseau.

Modification de l'ordre de précédence Dans la version originale, un pair désigné comme défaillant l'est de manière irrévocable. Ce comportement est dû à la règle de précédence suivante :

$$\forall i, j \in \mathbb{N}, \forall s \in \{Alive, Suspect\} \cdot \langle s, i\rangle < \langle Confirm, j\rangle$$

pour un noeud donné. Ainsi, un noeud déclaré comme défaillant par un autre noeud doit changer d'identité pour rejoindre de nouveau le groupe.

Ce choix n'est cependant pas anodin : il implique que la taille de la liste des collaborateur-rices croît de manière linéaire avec le nombre de connexions. S'agissant du paramètre avec le plus grand ordre de grandeur de l'application, nous avons cherché à le diminuer.

Nous avons donc modifié les règles de précédence de la manière suivante :

$$\forall i, j \in \mathbb{N}, i < j, \forall s, t \in \{Alive, Suspect, Confirm\} \cdot \langle i, s\rangle < \langle j, t\rangle$$

et

$$\forall i \in \mathbb{N} \cdot \langle i, Alive\rangle < \langle i, Suspect\rangle < \langle i, Confirm\rangle$$

Ces modifications permettent de donner la précédence au numéro d'incarnation, et d'utiliser le statut du collaborateur pour trancher seulement en cas d'égalité par rapport au numéro d'incarnation actuel. Ceci permet à un noeud auparavant déclaré comme défaillant de revenir dans le groupe en incrémentant son numéro d'incarnation. La taille de la liste des collaborateur-rices devient dès lors linéaire par rapport au nombre de noeuds.

Ces modifications n'ont pas d'impact sur la convergence des listes des collaborateur-rices des différents noeuds. Une étude approfondie reste néanmoins à effectuer pour déterminer si ces modifications ont un impact sur la vitesse à laquelle un noeud défaillant est déterminé comme tel par l'ensemble des noeuds.

Ajout d'un mécanisme de synchronisation La seconde modification que nous avons effectué concerne l'ajout d'un mécanisme de synchronisation entre pairs. En effet, le papier ne précise pas de procédure particulière lorsqu'un nouveau pair rejoint le réseau. Pour obtenir la liste des collaborateur-rices, ce dernier doit donc la demander à un autre pair.

Nous avons donc implémenté pour la liste des collaborateur-rices un mécanisme similaire à celui présenté en section 2.3.2 : à sa connexion, puis de manière périodique, un noeud envoie une requête de synchronisation à un noeud cible choisi de manière aléatoire. Ce message sert aussi à transmettre l'état courant du noeud source au noeud cible. En réponse, le noeud cible lui envoie l'état courant de sa liste. À la réception de cette dernière, le noeud source fusionne la liste reçue avec sa propre liste. Cette fusion conserve l'entrée la plus récente pour chaque noeud.

Pour récapituler, les mises à jour du groupe sont diffusées de manière atomique de façon épidémique, en utilisant les messages du mécanisme de détection des défaillances des noeuds. De manière additionnelle, un mécanisme d'anti-entropie permet à deux noeuds de synchroniser leur état. Ce mécanisme nous permet de pallier aux défaillances éventuelles du réseau. Ainsi, nous avons dans les faits mis en place un CRDT synchronisé par différences pour la liste des collaborateur-rices.

Synthèse

Pour générer et maintenir la liste des collaborateur-rices, nous avons implémenté le protocole distribué d'appartenance au réseau SWIM [33]. Par rapport à la version originale, nous avons procédé à plusieurs modifications, notamment pour gérer plus efficacement les reconnexions successives d'un même noeud.

Ainsi, nous avons implémenté un mécanisme dont la complexité spatiale dépend linéairement du nombre de noeuds. Sa complexité en temps et sa complexité en communication, elles, sont indépendantes de ce paramètre. Elles dépendent en effet de paramètres dont nous choisissons les valeurs : la fréquence de déclenchement du mécanisme de détection de défaillance et le nombre de mises à jour du groupe propagées par message.

Des améliorations au protocole SWIM furent proposées dans [34]. Ces modifications visent notamment à réduire le délai de détection d'un noeud défaillant, ainsi que réduire le taux de faux positifs. Ainsi, une perspective est d'implémenter ces améliorations dans MUTE.

2.3.4 Curseurs

Toujours dans le but d'offrir des fonctionnalités de conscience de groupe aux utilisateurs pour leur permettre de se coordonner aisément, nous avons implémenté dans MUTE l'affichage des curseurs distants.

Pour représenter fidèlement la position des curseurs des collaborateur-rices distants, nous nous reposons sur les identifiants du CRDT choisi pour représenter la séquence. Le fonctionnement est similaire à la gestion des modifications du document : lorsque l'éditeur indique que l'utilisateur a déplacé son curseur, nous récupérons son nouvel index. Nous recherchons ensuite l'identifiant correspondant à cet index dans la séquence répliquée et le diffusons aux collaborateur-rices.

À la réception de la position d'un curseur distant, nous récupérons l'index correspondant à cet identifiant dans la séquence répliquée et représentons un curseur à cet index. Il est intéressant de noter que si l'identifiant a été supprimé en concurrence, nous pouvons à la place récupérer l'index de l'élément précédent et ainsi indiquer à l'utilisateur où son collaborateur est actuellement en train de travailler.

De façon similaire, nous gérons les sélections de texte à l'aide de deux curseurs : un curseur de début et un curseur de fin de sélection.

2.4 Couche réseau

Pour permettre aux différents noeuds de communiquer, MUTE repose sur la librairie Netflux [9]. Développée au sein de l'équipe Coast, cette librairie permet de construire un réseau P2P entre des navigateurs, mais aussi des bots.

9. `https://github.com/coast-team/netflux`

2.4.1 Établissement d'un réseau P2P entre navigateurs

Pour créer un réseau P2P entre navigateurs, Netflux utilise la technologie Web Real-Time Communication (WebRTC). WebRTC est une API [10] de navigateur spécifiée en 2011, et en cours d'implémentation dans les différents navigateurs depuis 2013. Elle permet de créer une connexion directe entre deux navigateurs pour échanger des médias audio et/ou vidéo, ou simplement des données.

Cette API utilise pour cela un ensemble de protocoles. Ces protocoles réintroduisent des serveurs dans l'architecture système de MUTE. Dans la Figure 2.10, nous représentons un réseau P2P créé avec WebRTC et les différents serveurs impliqués.

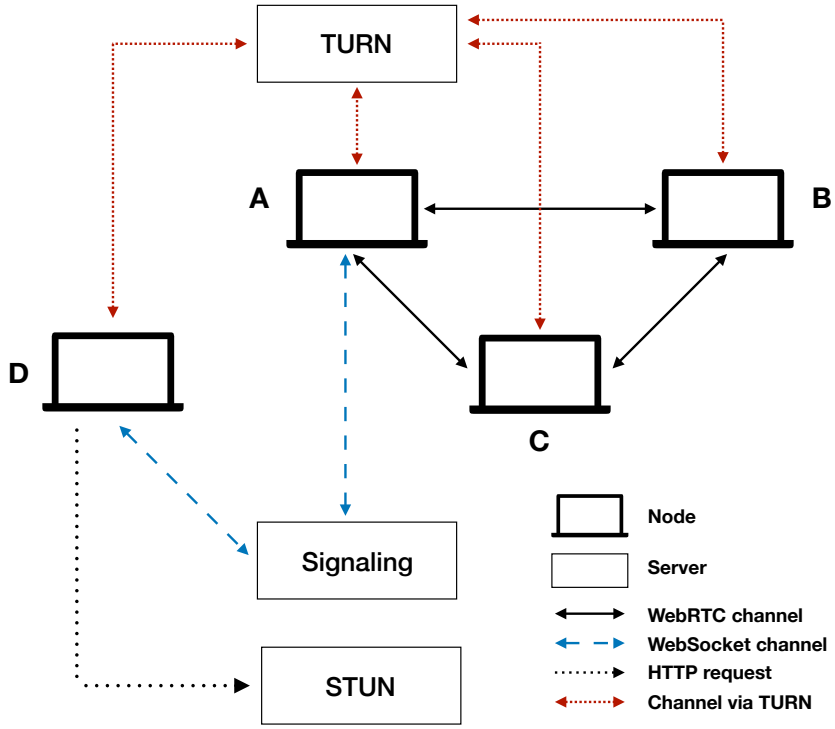


FIGURE 2.10 – Architecture système pour la couche réseau de MUTE

Nous décrivons ci-dessous leur rôle respectif dans la collaboration.

Serveur de signalisation

Pour rejoindre un réseau P2P déjà établi, un nouveau noeud a besoin de découvrir les noeuds déjà connectés et de pouvoir communiquer avec eux. Le serveur de signalisation offre ces fonctionnalités.

Au moins un noeud du réseau P2P doit maintenir une connexion avec le serveur de signalisation. À sa connexion, un nouveau noeud contacte le serveur de signalisation. Il est mis en relation avec un noeud du réseau P2P par son intermédiaire et échange les

10. Application Programming Interface (API) : Interface de Programmation

différents messages de WebRTC nécessaires à l'établissement d'une connexion P2P entre eux.

Une fois cette première connexion P2P établie, le nouveau noeud contacte et communique avec les autres noeuds par l'intermédiaire du premier noeud. Il peut alors terminer sa connexion avec le serveur de signalisation.

Serveur STUN

Pour se connecter, les noeuds doivent s'échanger plusieurs informations logicielles et matérielles, notamment leur adresse IP publique respective. Cependant, un noeud n'a pas accès à cette donnée lorsque son routeur utilise le protocole NAT. Le noeud doit alors la récupérer.

Pour permettre aux noeuds de découvrir leur adresse IP publique, WebRTC repose sur le protocole STUN. Ce protocole consiste simplement à contacter un serveur tiers dédié à cet effet. Ce serveur retourne en réponse au noeud qui le contacte son adresse IP publique.

Serveur TURN

Il est possible que des noeuds provenant de réseaux différents ne puissent établir une connexion P2P directe entre eux, par exemple à cause de restrictions imposées par leur pare-feux respectifs. Pour contourner ce cas de figure, WebRTC utilise le protocole TURN.

Ce protocole consiste à utiliser un serveur tiers comme relais entre les noeuds. Ainsi, les noeuds peuvent communiquer par son intermédiaire tout au long de la collaboration. Les échanges sont chiffrés, afin que le serveur TURN ne représente pas une faille de sécurité.

Rôle des serveurs

Ainsi, WebRTC implique l'utilisation de plusieurs serveurs.

Les serveurs de signalisation et STUN sont nécessaires pour permettre à de nouveaux noeuds de rejoindre la collaboration. Autrement dit, leur rôle est ponctuel : une fois le réseau P2P établit, les noeuds n'ont plus besoin d'eux. Ces serveurs peuvent alors être coupés sans impacter la collaboration.

À l'inverse, les serveurs TURN jouent un rôle plus prédominant dans la collaboration. Ils sont nécessaires dès lors que des noeuds proviennent de réseaux différents et sont alors requis tout au long de la collaboration. Une panne de ces derniers entraverait la collaboration puisqu'elle résulterait en une partition des noeuds. Il est donc primordial de s'assurer de la disponibilité et fiabilité de ces serveurs.

2.4.2 Topologie réseau

Netflux établit un réseau P2P par document. Chaque réseau P2P est un réseau entièrement maillé : chaque noeud se connecte à l'ensemble des autres noeuds.

Cette topologie simple est adaptée à des groupes de petite taille, mais ne passe pas à l'échelle. D'autres topologies limitant le nombre de connexions par noeuds, telle que celle décrite par [35], pourraient être implémentées pour adresser cette limite.

2.5 Couche sécurité

La couche sécurité a pour but de garantir l'authenticité et la confidentialité des messages échangés par les noeuds. Pour cela, elle implémente un mécanisme de chiffrement de bout en bout.

Pour chiffrer les messages, MUTE utilise un mécanisme de chiffrement à base de clé de groupe. Le protocole choisi est le protocole Burmester-Desmedt [36]. Il nécessite que chaque noeud possède une paire de clés de chiffrement et enregistre sa clé publique auprès d'un PKI[11].

Afin d'éviter qu'un PKI malicieux n'effectue une attaque de l'homme au milieu sur la collaboration, les noeuds doivent vérifier le bon comportement des PKI de manière non-coordonnée. À cet effet, MUTE implémente le mécanisme d'audit de PKI Trusternity [31, 32]. Son fonctionnement nécessite l'utilisation d'un registre publique sécurisé *append-only*, c.-à-d. une blockchain.

L'architecture système nécessaire pour la couche sécurité est présentée dans la Figure 2.11.

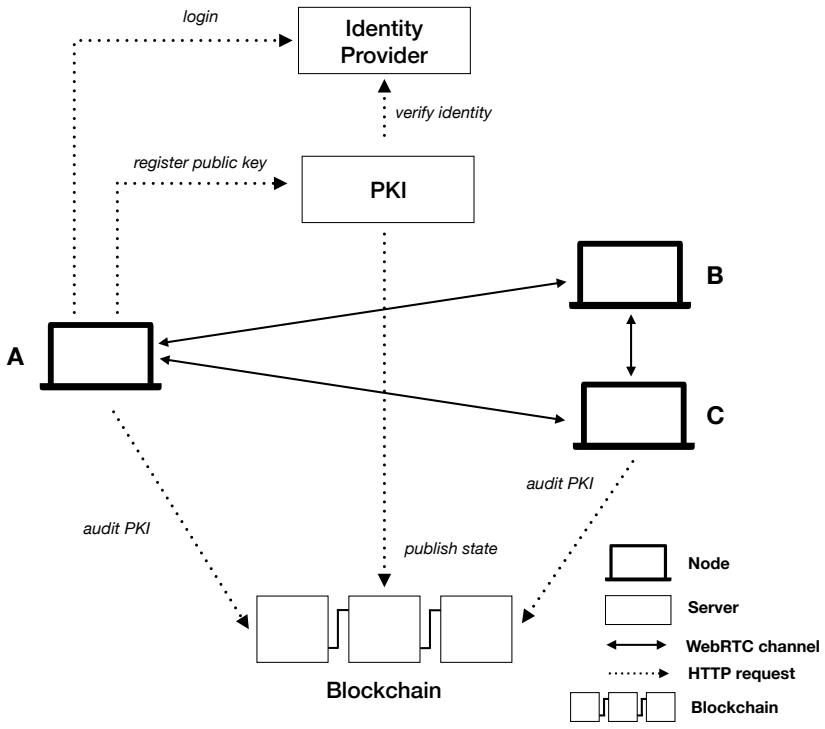


FIGURE 2.11 – Architecture système pour la couche sécurité de MUTE

Cette couche sécurité s'ajoute au mécanisme de chiffrement des messages inhérent à WebRTC. Cela nous offre de nouvelles possibilités : plutôt que de créer un réseau P2P par document, nous pouvons désormais mettre en place un réseau P2P global. Les messages étant chiffrés de bout en bout, les noeuds peuvent communiquer en toute sécurité

11. Public Key Infrastructure (PKI) : Infrastructure de gestion de clés

et confidentialité par l'intermédiaire de noeuds tiers, c.-à-d. des noeuds extérieurs à la collaboration.

Une limite de l'approche actuelle est que la clé de groupe change avec l'évolution des noeuds connectés : à chaque connexion ou déconnexion d'un noeud, une nouvelle clé est recalculée avec les collaborateur-rices présents. Cette évolution fréquente de la clé de chiffrement, nécessaire pour garantir la *backward secrecy* et *forward secrecy*, nous empêche par exemple de stocker les opérations de manière chiffrée chez des noeuds tiers. Cette fonctionnalité serait cependant bien pratique pour permettre à un noeud de récupérer la dernière version de ses documents, même en l'absence de ses collaborateur-rices. Une autre clé de chiffrement, dédiée au stockage, devrait être mise en place, ainsi qu'un mécanisme de découverte des noeuds tiers stockant les données de la collaboration.

2.6 Conclusion

Dans ce chapitre, nous avons présenté Multi User Text Editor (MUTE), notre éditeur collaboratif temps réel P2P chiffré de bout en bout.

MUTE permet d'éditer de manière collaborative des documents texte. Pour représenter les documents, MUTE implémente les structures de données répliquées décrites dans la **??** et le **??**. Ces CRDTs offrent de nouvelles méthodes de collaborer, notamment en permettant de collaborer de manière synchrone ou asynchrone de manière transparente.

Pour permettre aux noeuds de communiquer, MUTE utilise WebRTC. Cette technologie permet de construire un réseau P2P entre navigateurs. Plusieurs serveurs sont néanmoins requis, notamment pour la découverte des pairs et pour la communication entre des noeuds dont les pare-feux respectifs empêche l'établissement d'une connexion directe.

Finalement, MUTE implémente un mécanisme de chiffrement de bout en bout garantissant l'authenticité et la confidentialité des échanges entre les noeuds. Ce mécanisme reposant sur d'autres serveurs, les PKIs, MUTE intègre un mécanisme d'audit permettant de détecter leurs éventuels comportements malicieux.

Chapitre 3

Conclusions et perspectives

Sommaire

Dans ce chapitre, nous revenons sur les contributions présentés dans cette thèse. Nous rappelons le contexte dans lequel elles s'inscrivent, récapitulons leurs spécificités et apports, et finalement présentons leurs limites que nous identifions. Puis, nous concluons ce manuscrit en présentant plusieurs pistes de recherche qui nous restent à explorer à l'issue de cette thèse. Les premières s'inscrivent dans la continuité directe de nos travaux sur un mécanisme de ré-identification pour CRDTs pour Séquence dans un système P2P à large échelle sujet au churn. Les dernières traduisent quant à elles notre volonté de recentrer nos travaux sur le domaine plus général des CRDTs.

3.1 Résumés des contributions

3.1.1 Réflexions sur l'état de l'art des CRDTs

Les Conflict-free Replicated Data Types (CRDTs) [22] sont de nouvelles spécifications des types de données. Ils sont conçus pour permettre à un ensemble de noeuds d'un système de répliquer une même donnée et pour leur permettre de la consulter et de la modifier sans aucune coordination préalable. Les noeuds se synchronisent

L'absence de coordination entre les noeuds avant modifications implique que des noeuds peuvent modifier la donnée en concurrence. De telles modifications peuvent donner lieu à des conflits, e.g. l'ajout et la suppression en concurrence d'un même élément dans un ensemble. Pour pallier ce problème, les CRDTs incorporent un mécanisme de résolution de conflits automatiques directement au sein de leur spécification.

Il convient de noter qu'il existe plusieurs solutions possibles pour résoudre un conflit. Pour reprendre l'exemple de l'élément ajouté et supprimé en concurrence d'un ensemble, nous pouvons par exemple soit le conserver l'élément, soit le supprimer. Nous parlons alors de sémantique du mécanisme de résolution de conflits automatique.

De la même manière, il existe plusieurs approches possibles pour synchroniser les noeuds, e.g. diffuser chaque modification de manière atomique ou diffuser l'entièreté de l'état périodiquement. Ainsi, lors de la définition d'un CRDT, il convient de préciser les sémantiques de résolution de conflits qu'il adopte et le modèle de synchronisation qu'il utilise [54].

Depuis leur formalisation, les travaux sur les CRDTs ont abouti à la conception de nouveaux, soit en spécifiant de nouvelles sémantiques de résolution de conflits pour un type de données [55], soit en spécifiant de nouveaux modèles de synchronisation [56] ou en enrichissant les spécifications des modèles existants [57, 58].

Dans notre présentation des CRDTs (cf. **??**, page **??**), nous présentons chacun de ces aspects. Cependant, nous nous ne limitons pas à retranscrire l'état de l'art de la littérature. Notamment au sujet du modèle de synchronisation par opérations, nous précisons que le modèle de livraison causal n'est pas nécessaire pour l'ensemble des CRDTs synchronisés par opérations, c.-à-d. que certains peuvent adopter des modèles de livraison moins contraints et donc moins coûteux. Cette précision nous permet de proposer une étude comparative des différents modèles de synchronisation qui est, à notre connaissance, l'une des plus précises de la littérature (cf. **??**, page **??**).

Nous présentons ensuite les différents CRDTs pour le type Séquence de la littérature (cf. **??**, page **??**). Nous mettons alors en exergue les deux approches proposées pour concevoir le mécanisme de résolution de conflits automatiques pour le type Séquence, c.-à-d. l'approche à pierres tombales et l'approche à identifiants densément ordonnés. De nouveau, cette rétrospective nous permet d'expliciter des angles morts des articles d'origine, notamment vis-à-vis des modèle de livraison des opérations des CRDTs proposés. Puis, nous mettons en lumière les limites des évaluations comparant les deux approches, c.-à-d. le couplage entretenu entre approche du mécanisme de résolution de conflits et choix d'implémentations. Cette limite empêche d'établir la supériorité d'une des approches par rapport à l'autre. Finalement, nous conjecturons que le surcoût de ces deux approches est le même, c.-à-d. le coût nécessaire à la représentation d'un espace dense. Nous précisons dès lors par le biais de notre propre étude comparative comment ce surcoût s'exprime dans chacune des approches , c.-à-d. le compromis entre surcoût en métadonnées, calculs et bande-passante proposé par les deux approches (cf. **??**, page **??**).

Ces réflexions que nous présentons sur l'état des CRDTs définissent plusieurs pistes de recherches. Une première d'entre elles concerne notre étude comparative des modèles de

synchronisation. D'après les critères que nous utilisons, une conclusion possible de cette comparaison est que le modèle de synchronisation par différences d'états rend obsolètes les modèles de synchronisation par états et par opérations. En effet, le modèle de synchronisation par différences d'états apparaît comme adapté à l'ensemble des contextes d'utilisation qui étaient jusqu'à lors exclusifs à ces derniers, de par les multiples stratégies qu'il permet, e.g. synchronisation par états complets, synchronisation par états irréductibles...

Cette conclusion nous paraît cependant hâtive. Il convient d'étendre notre étude comparative pour prendre en compte des critères de comparaison additionnels pour confirmer cette conjecture, ou l'invalider et définir plus précisément les spécificités de chacun des modèles de synchronisation. Nous détaillons cette piste de recherche dans la sous-section 3.2.3.

Une seconde piste de recherche possible concerne les deux approches utilisées pour concevoir le mécanisme de résolution de conflits des CRDTs pour le type Séquence. Comme dit précédemment, nous conjecturons que ces deux approches ne sont finalement que deux manières différentes de représenter une même information : la position d'un élément dans un espace dense. La différence entre ces approches résiderait uniquement dans la manière que chaque représentation influe sur les performances du CRDT. Une piste de travail serait donc de confirmer cette conjecture, en proposant une formalisation unique des CRDTs pour le type Séquence.

3.1.2 Ré-identification sans coordination pour les CRDTs pour Séquence

Pour privilégier leur disponibilité, latence et tolérance aux pannes, les systèmes distribués peuvent adopter le paradigme de la réplication optimiste [16]. Ce paradigme consiste à relaxer la cohérence de données entre les noeuds du système pour leur permettre de consulter et modifier leur copie locale sans se coordonner. Leur copies peuvent alors temporairement diverger avant de converger de nouveau une fois les modifications de chacun propagées. Cependant, cette approche nécessite l'emploi d'un mécanisme de résolution de conflits pour assurer la convergence même en cas de modifications concurrentes. Pour cela, l'approche des CRDTs [21, 22] propose d'utiliser des types de données dont les modifications sont nativement commutatives.

Depuis la spécification des CRDTs, la littérature a proposé plusieurs de ces mécanismes résolution de conflits automatiques pour le type de données Séquence [59, 60, 61, 62]. Cependant, ces approches souffrent toutes d'un surcoût croissant de manière monotone. Ce problème a été identifié par la communauté, et celle-ci a proposé pour y répondre des mécanismes permettant soit de réduire la croissance du surcoût [63, 64], soit d'effectuer une GC du surcoût [60, 23, 24]. Nous avons cependant déterminé que ces mécanismes ne sont pas adaptés aux systèmes P2P à large échelle souffrant de churn et utilisant des CRDTs pour Séquence à granularité variable.

Dans le cadre de cette thèse, nous avons donc souhaité proposer un nouveau mécanisme adapté à ce type de systèmes. Pour cela, nous avons suivi l'approche proposée par [23, 24] : l'utilisation d'un mécanisme pour ré-assigner de nouveaux identifiants aux élements stockées dans la séquence. Nous avons donc proposé un nouveau mécanisme appartenant

à cette approche pour le CRDT LogootSplit [28].

Notre proposition prend la forme d'un nouvel CRDT pour Séquence à granularité variable : RenamableLogootSplit. Ce nouveau CRDT associe à LogootSplit un nouveau type de modification, *ren*, permettant de produire une nouvelle séquence équivalente à son état précédent. Cette nouvelle modification tire profit de la granularité variable de la séquence pour produire un état de taille minimale : elle assigne à tous les éléments des identifiants de position issus d'un même intervalle. Ceci nous permet de minimiser les métadonnées que la séquence doit stocker de manière effective.

Afin de gérer les opérations concurrentes aux opérations *ren*, nous définissons pour ces dernières un algorithme de transformation. Pour cela, nous définissons un mécanisme d'époques nous permettant d'identifier la concurrence entre opérations. De plus, nous introduisons une relation d'ordre strict total, *priority*, pour résoudre de manière déterministe le conflit provoqué par deux opérations *ren*, c.-à-d. pour déterminer quelle opération *ren* privilégier. Finalement, nous définissons deux algorithmes, `renameId` et `revertRenameId`, qui permettent de transformer les opérations concurrentes à une opération *ren* pour prendre en compte l'effet de cette dernière. Ainsi, notre algorithme permet de détecter et de transformer les opérations concurrentes aux opérations *ren*, sans nécessiter une coordination synchrone entre les noeuds.

Pour valider notre approche, nous proposons une évaluation expérimentale de cette dernière. Cette évaluation se base sur des traces de sessions d'édition collaborative que nous avons généré par simulations. Chacune de ces simulations représente la rédaction collaborative d'un document texte par 10 noeuds.

Notre évaluation nous permet de valider de manière empirique les résultats attendus. Le premier d'entre eux concerne la convergence des noeuds. En effet, nos simulations nous ont permis de valider que l'ensemble des noeuds obtenaient des états finaux équivalents, même en cas d'opérations *ren* concurrentes.

Notre évaluation nous a aussi permis de valider que le mécanisme de renommage réduit à une taille minimale le surcoût du mécanisme de résolution de conflits incorporé dans le CRDT pour Séquence.

L'évaluation expérimentale nous a aussi permis de prendre conscience d'effets additionnels du mécanisme de renommage que nous n'avions pas anticipé. Notamment, elle montre que le surcoût éventuel du mécanisme de renommage, notamment en termes de calculs, est toutefois contrebalancé par l'amélioration précisée précédemment, c.-à-d. la réduction de la taille de la séquence.

Finalement, notons que le mécanisme que nous proposons est partiellement générique : il peut être adapté à d'autres CRDTs pour Séquence à granularité variable, e.g. un CRDT pour Séquence appartenant à l'approche à pierres tombales. Dans le cadre d'une telle démarche, nous pourrions réutiliser le système d'époques, la relation *priority* et l'algorithme de contrôle qui identifie les transformations à effectuer. Pour compléter une telle adaptation, nous devrions cependant concevoir de nouveaux algorithmes `renameId` et `revertRenameId` spécifiques et adaptés au CRDT choisi.

Le mécanisme de renommage que nous présentons souffre néanmoins de plusieurs limites. La première d'entre elles concerne ses performances. En effet, notre évaluation expérimentale a mis en lumière le coût important en l'état de la modification *ren* par rapport aux autres modifications en termes de calculs (cf. **??**, page **??**). De plus, chaque opération *ren* comporte une représentation de l'ancien état qui doit être maintenue par les noeuds jusqu'à leur stabilité causale. Le surcoût en métadonnées introduit par un ensemble d'opérations *ren* concurrentes peut donc s'avérer important, voire pénalisant (cf. **??**, page **??**). Pour répondre à ces problèmes, nous identifions trois axes d'amélioration :

(i) La définition de stratégies de déclenchement du renommage efficaces. Le but de ces stratégies serait de déclencher le mécanisme de renommage de manière fréquente, de façon à garder son temps d'exécution acceptable, mais tout visant à minimiser la probabilité que les noeuds produisent des opérations *ren* concurrentes, de façon à minimiser le surcoût en métadonnées.

(ii) La définition de relations *priority* efficaces. Nous développons ce point dans la sous-section 3.2.1.

(iii) La proposition d'algorithmes de renommage efficaces. Cette amélioration peut prendre la forme de nouveaux algorithmes pour `renameId` et `revertRenameId` offrant une meilleure complexité en temps. Il peut aussi s'agir de la conception d'une nouvelle approche pour renommer l'état et gérer les modifications concurrentes, e.g. un mécanisme de renommage basé sur le journal des opérations (cf. **??**, page **??**).

Une seconde limite de RenamableLogootSplit que nous identifions concerne son mécanisme de GC des métadonnées introduites par le mécanisme de renommage. En effet, pour fonctionner, ce dernier repose sur la stabilité causale des opérations *ren*. Pour rappel, la stabilité causale représente le contexte causal commun à l'ensemble des noeuds du système. Pour le déterminer, chaque noeud doit récupérer le contexte causal de l'ensemble des noeuds du système. Ainsi, l'utilisation de la stabilité causale comme pré-requis pour la GC de métadonnées constitue une contrainte forte, voire prohibitive, dans les systèmes P2P à large échelle sujet au churn. En effet, un noeud du système déconnecté de manière définitive suffit pour empêcher la stabilité causale de progresser, son contexte causal étant alors indéterminé du point de vue des autres noeuds. Il s'agit toutefois d'une limite récurrente des mécanismes de GC distribués et asynchrones [60, 57, 65]. Nous présentons une piste de travail possible pour pallier ce problème dans la sous-section 3.2.2.

3.1.3 Éditeur de texte collaboratif P2P chiffré de bout en bout

Les applications collaboratives permettent à des utilisateur-rices de réaliser collaborativement une tâche. Elles permettent à plusieurs utilisateur-rices de consulter la version actuelle du document, de la modifier et de partager leurs modifications avec les autres. Ceci permet de mettre en place une réflexion de groupe, ce qui améliore la qualité du résultat produit [39, 7].

Cependant, les applications collaboratives sont historiquement des applications centralisées, e.g. Google Docs [25]. Ce type d'architecture induit des défauts d'un point de vue technique, e.g. faible capacité de passage à l'échelle et faible tolérance aux pannes,

mais aussi d'un point de vue utilisateur, e.g. perte de la souveraineté des données et absence de garantie de pérennité.

Les travaux de l'équipe Coast s'inscrivent dans une mouvance souhaitant résoudre ces problèmes et qui a conduit à la définition d'un nouveau paradigme d'applications : les LFS [14]. Le but de ce paradigme est la conception d'applications collaboratives, P2P, pérennes et rendant la souveraineté de leurs données aux utilisateur-rices.

Dans le cadre de cette démarche, l'équipe Coast développe depuis plusieurs années l'application Multi User Text Editor (MUTE), un éditeur de texte web collaboratif P2P temps réel chiffré de bout en bout. Cette application sert à la fois de plateforme de démonstration et de recherche pour les travaux de l'équipe, mais aussi de PoC pour les LFS.

Dans le cadre de cette thèse, nous avons implémenté dans MUTE nos travaux de recherche portant sur le nouvel CRDT pour le type Séquence : RenamableLogootSplit. MUTE a aussi servi à l'équipe pour présenter ses travaux concernant l'authentification des utilisateur-rices dans un système P2P [32]. Finalement, MUTE nous a permis de nous d'étudier et/ou de présenter les travaux de recherche existants concernant :

(i) Les protocoles distribués d'appartenance au groupe [33].

(ii) Les mécanismes d'anti-entropie [53].

(iii) Les protocoles d'établissement de clés de chiffrement de groupe [36].

(iv) Les protocoles d'établissement de topologies réseaux efficientes [35].

(v) Les mécanismes de conscience de groupe.

MUTE offre donc, à notre connaissance, le tour d'horizon le plus complet des travaux de recherche permettant la conception d'applications LFS. Cependant, cela ne dispense pas MUTE de souffrir de plusieurs limites.

Tout d'abord, l'environnement web implique un certain nombre de contraintes, notamment au niveau des technologies et protocoles disponibles. Notamment, le protocole WebRTC repose sur l'utilisation de serveurs de signalisation, c.-à-d. de points de rendez-vous des pairs, et de serveurs de relai, c.-à-d. d'intermédiaires pour communiquer entre pairs lorsque les configurations de leur réseaux respectifs interdisent l'établissement d'une connexion directe. Ainsi, les applications P2P web doivent soit déployer et maintenir leur propre infrastructure de serveurs, soit reposer sur une infrastructure existante, e.g. celle proposée par OpenRelay [66]. Afin de minimiser l'effort requis aux applications P2P et la confiance exigée à leurs utilisateur-rices, nous devons supporter la mise en place d'une telle infrastructure transparente et pérenne.

Une autre limite de ce système que nous identifions concerne l'utilisabilité des systèmes P2P de manière générale. L'expérience vécue suivante constitue à notre avis un exemple éloquent des limites actuelles de l'application MUTE dans ce domaine. Après avoir rédigé une version initiale d'un document, nous avons envoyé le lien du document à notre collaborateur pour relecture et validation. Lorsque notre collaborateur a souhaité accéder au document, celui-ci s'est retrouvé devant une page blanche : comme nous nous étions déconnecté du système entretemps, c.-à-d. plus aucun pair n'était disponible pour

effectuer une synchronisation. Notre collaborateur était donc dans l'incapacité de récupérer l'état et d'effectuer sa tâche. Afin de pallier ce problème, une solution possible est de faire reposer MUTE sur un réseau P2P global, e.g. le réseau de InterPlanetary File System (IPFS) [67], et d'utiliser les pairs de ce dernier, potentiellement des pairs étrangers à l'application, comme pairs de stockage pour permettre une synchronisation future. Cette solution limite ainsi le risque qu'un pair ne puisse récupérer l'état du document faute de pairs disponibles. Cependant, elle nécessite de mettre en place un mécanisme de réplication de données additionnel. Ce mécanisme de réplication sur des pairs additionnels doit cependant garantir qu'il n'introduit pas de vulnérabilités, e.g. la possibilité pour les pairs de stockage selectionnés de reconstruire et consulter le document.

3.2 Perspectives

3.2.1 Définition de relations de priorité pour minimiser les traitements

Dans la **??**, nous avons spécifié la relation *priority* (cf. **??**, page **??**). Pour rappel, cette relation doit établir un ordre strict total sur les époques de notre mécanisme de renommage.

Cette relation nous permet ainsi de résoudre le conflit provoqué par la génération de modifications *ren* concurrentes en les ordonnant. Grâce à cette relation relation d'ordre, les noeuds peuvent 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) entre les époques courantes des noeuds. Cela permet aux noeuds d'appliquer le mécanisme de GC pour supprimer les époques devenues obsolètes et leur anciens états associés, pour ainsi 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. cette approche n'introduit pas de métadonnées supplémentaires à stocker et diffuser, et l'algorithme de comparaison utilisé est 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 reciprocité.

Cette définition présente cependant plusieurs limites. La limite que nous identifions est sa décorrélation avec le coût et le 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 ainsi 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. De plus, la transition de l'ensemble des noeuds depuis leur époque courante respective vers cette nouvelle époque cible induit un coût en calculs, potentiellement important (cf. **??**, page **??**).

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'adjoindre 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, telle qu'une 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.

Finalement, il sera nécessaire de valider l'approche proposée par une évaluation comparative par rapport à l'approche actuelle. Cette évaluation pourrait consister à monitorer le coût du système pour observer si l'approche proposée permet de réduire les calculs de manière globale. Plusieurs configurations de paramètres pourraient aussi être utilisées pour déterminer l'impact respectif de chaque paramètre sur les résultats.

3.2.2 Détection et fusion manuelle de versions distantes

À l'issue de cette thèse, nous identifions deux limites des mécanismes de résolution de conflits automatiques dans les systèmes P2P à large échelle sujets au churn :

(i) La croissance continue de leur surcoût dûe à l'utilisation du contexte causal.

(ii) Leur compréhension limitée des intentions des utilisateur-rices, ce qui peut conduire à la génération d'anomalies lors de l'intégration de modifications.

Nous conjecturons que la probabilité que l'intégration de modifications résulte en des anomalies et que le travail nécessaire par les utilisateur-rices pour corriger ces anomalies

augmentent avec la distance entre la version de génération des modifications et la version d'intégration de ces dernières. Si cette conjecture se vérifie, nous proposons d'identifier la distance seuil entre versions à partir de laquelle l'utilisation d'un mécanisme de résolution de conflit automatique donné s'avère inefficace. Dès lors, nous proposons de recourir à un mécanisme d'intégration manuelle des modifications lorsque des modifications reçues sont originaires d'une version du document se trouvant au-delà de ce seuil.

Le recours à un mécanisme d'intégration manuelle des modifications permettrait alors :

(i) Prévenir la génération d'anomalies entravant la collaboration.

(ii) Supprimer les métadonnées nécessaires pour intégrer automatiquement des modifications originaires d'une version au-delà de la distance seuil, celles-ci n'étant dès lors plus requises puisque ces modifications seront intégrées manuellement. Par exemple, dans le cadre de RenamableLogootSplit, ce mécanisme nous permettrait de supprimer les époques se trouvant au-delà de cette distance, et ainsi leurs métadonnées associées.

Ainsi, cette approche nous permettrait de répondre aux deux problèmes identifiés précédemment.

Pour expliciter cette approche, revenons sur l'utilisation du contexte causal par les mécanismes de résolution de conflits automatiques. Le contexte causal est utilisé par les mécanismes de résolution de conflits automatiques pour :

(i) Satisfaire le modèle de cohérence causale, c.-à-d. assurer que si nous avons deux modifications m_1 et m_2 telles que $m1 \rightarrow m_2$, alors l'effet de m_2 supplantera celui de m_1. Ceci permet d'éviter des anomalies de comportement de la part de la structure de données du point de vue des utilisateur-rices, par exemple la résurgence d'un élément supprimé au préalable.

(ii) Permettre de préserver l'intention d'une modification malgré l'intégration préalable de modifications concurrentes.

Les mécanismes de résolution de conflits automatiques maintiennent donc le contexte causal de l'état de la donnée. Le contexte causal peut être représenté de différentes manières. Par exemple, le contexte causal peut prendre la forme du journal des opérations auquel on associe soit le vecteur de version correspondant à l'état de la donnée [68, 69], soit l'ensemble des extremités du Directed Acyclic Graph (DAG) formé par les opérations [70]. Cependant, de manière intrinsèque, le contexte causal ne fait que de croître au fur et à mesure que des modifications sont effectuées ou que des noeuds rejoignent le système, incrémentant son surcoût en métadonnées, calculs et bande-passante.

La stabilité causale permet cependant de réduire le surcoût lié au contexte causal. En effet, la stabilité causale permet d'établir le contexte commun à l'ensemble des noeuds, c.-à-d. l'ensemble des modifications que l'ensemble des noeuds ont intégré. Ces modifications font alors partie de l'histoire commune et n'ont plus besoin d'être considérées par les mécanismes de résolution de conflits automatiques. La stabilité causale permet donc de déterminer et de tronquer la partie commune du contexte causal pour éviter que ce dernier ne pénalise les performances du système à terme.

La stabilité causale est cependant une contrainte forte dans les systèmes P2P dynamiques à large échelle sujet au churn. Il ne suffit en effet que d'un noeud déconnecté pour empêcher la stabilité causale de progresser. Pour répondre à ce problème, nous avons dès lors tout un spectre d'approches possibles, proposant chacune un compromis entre le surcoût du contexte causal et la probabilité de rejeter des modifications. Les extremités de ce spectre d'approches sont les suivantes :

(i) Considérer tout noeud déconnecté comme déconnecté de manière définitive, et donc les ignorer dans le calcul de la stabilité causale. Cette première approche permet à la stabilité causale de progresser, et ainsi aux noeuds connectés de travailler dans des conditions optimales. Mais elle implique cependant que les modifications potentielles des noeuds déconnectés soient perdues, c.-à-d. de ne plus pouvoir les intégrer en l'absence d'un lien entre leur contexte causal de génération et le contexte causal actuel de chaque autre noeud. Il s'agit là de la stratégie la plus aggressive en terme de GC du contexte causal.

(ii) Assurer en toutes circonstances la capacité d'intégration des modifications des noeuds, même ceux déconnectés. Cette seconde approche permet de garantir que les modifications potentielles des noeuds déconnectés pourront être intégrées automatiquement, dans l'éventualité où ces derniers se reconnectent à terme. Mais elle implique de bloquer potentiellement de manière définitive la stabilité causale et donc le mécanisme de GC du contexte causal. Il s'agit là de la stratégie la plus timide en terme de GC du contexte causal.

La seconde limite que nous constatons est la limite des mécanismes actuels de résolution de conflits automatiques pour préserver l'intention des utilisateur-rices. Par exemple, les mécanismes de résolution de conflits automatiques pour le type Séquence présentés dans ce manuscrit (cf. **??**, page **??**) définissent l'intention de la manière suivante : *l'intégration de la modification par les noeuds distants doit reproduire l'effet de la modification sur la copie d'origine*. Cette définition assure que chaque modification est porteuse d'une intention, mais limite voire ignore toute la dimension sémantique de la dite intention. Nous conjecturons que l'absence de dimension sémantique réduit les cas d'utilisation de ces mécanismes.

Considérons par exemple une édition collaborative d'un même texte par un ensemble de noeuds. Lors de la présence d'une faute de frappe dans le texte, e.g. le mot "HLLO", plusieurs utilisateur-rices peuvent la corriger en concurrence, c.-à-d. insérer l'élément "E" entre "H" et "L". Les mécanismes de résolution de conflits automatiques permettent aux noeuds d'obtenir des résultats qui convergent mais à notre sens insatisfaisant, e.g. "HEEEEEELLO". Nous considérons ce type de résultats comme des anomalies, au même titre que l'entrelacement [71]. Dans le cadre de collaborations temps réel à échelle limitée, nous conjecturons cependant qu'une granularité fine des modifications permet de pallier ce problème. En effet, les utilisateur-rices peuvent observer une anomalie produite par le mécanisme de résolution de conflits automatique, déduire l'intention initiale des modifications concernées et la restaurer par le biais d'actions supplémentaires de compensation.

Cependant, dans le cadre de collaborations asynchrones ou à large échelle, nous conjecturons que ces anomalies de résolution de conflits s'accumulent. Cette accumulation peut atteindre un seuil rendant laborieuse la déduction et le rétablissement de l'intention ini-

tiale des modifications. Le travail imposé aux utilisateur-rices pour résoudre ces anomalies par le biais d'actions de compensation peut alors entraver la collaboration. Pour reprendre l'exemple de l'édition collaborative de texte, nous pouvons constater de tels cas suite à de la duplication de contenu et/ou l'entrelacement de mots, phrases voire paragraphes nuisant à la clarté et correction du texte. Il convient alors de s'interroger sur le bien-fondé de l'utilisation de mécanismes de résolutions de conflits automatiques pour intégrer un ensemble de modifications dans l'ensemble des situations.

Ainsi, pour répondre aux limites des mécanismes de résolution conflits automatiques dans les systèmes P2P à large échelle, c.-à-d. l'augmentation de leur surcoût et la pertinence de leur résultat, nous souhaitons proposer une approche combinant un ou des mécanismes de résolution de conflits automatiques avec un ou des mécanismes de résolution de conflits manuels. L'idée derrière cette approche est de faire varier le mécanisme de résolution de conflits utilisé pour intégrer des modifications. Le choix du mécanisme de résolution de conflits utilisé peut se faire à partir de la valeur d'une distance calculée entre la version courante de la donnée répliquée et celle de la génération de la modification à intégrer, ou d'une évaluation de la qualité du résultat de l'intégration de la modification. Par exemple :

(i) Si la distance calculée se trouve dans un intervalle de valeurs pour lequel nous disposons d'un mécanisme de résolution de conflits automatique satisfaisant, utiliser ce dernier. Ainsi, nous pouvons envisager de reposer sur plusieurs mécanismes de résolution de conflits automatiques, de plus en plus complexes et pertinents mais coûteux, sans dégrader les performances du système dans le cas de base.

(ii) Si la distance calculée dépasse la distance seuil, c.-à-d. que nous ne disposons plus à ce stade de mécanismes de résolution de conflits automatiques satisfaisants, faire intervenir les utilisateur-rices par le biais d'un mécanisme de résolution de conflits manuel. L'utilisation d'un mécanisme manuel n'exclut cependant pas tout pré-travail de notre part pour réduire la charge de travail des utilisateur-rices dans le processus de fusion.

Dans un premier temps, cette approche pourrait se focaliser sur un type d'application spécifique, e.g. l'édition collaborative de texte.

Cette approche nous permettrait de répondre aux limites soulevées précédemment. En effet, elle permettrait de limiter la génération d'anomalies par le mécanisme de résolution de conflits automatique en faisant intervenir les utilisateur-rices. Puis, puisque nous déléguons aux utilisateur-rices l'intégration des modifications à partir d'une distance seuil, nous pouvons dès lors reconsidérer les métadonnées conservées par les noeuds pour les mécanismes de résolution de conflits automatiques. Notamment, nous pouvons identifier les noeuds se trouvant au-delà de cette distance seuil d'après leur dernier contexte causal connu et ne plus les prendre en compte pour le calcul de la stabilité causale. Cette approche permettrait donc de réduire le surcoût lié au contexte causal et limiter la perte de modifications, tout en prenant en considération l'ajout de travail aux utilisateur-rices.

Pour mener à bien ce travail, il conviendra tout d'abord de définir la notion de distance entre versions de la donnée répliquée. Nous envisageons de baser cette dernière sur les deux

aspects temporel et spatial, c.-à-d. en utilisant la distance entre contextes causaux et la distance entre contenus. Dans le cadre de l'édition collaborative, nous pourrons pour cela nous baser sur les travaux existants pour évaluer la distance entre deux textes. *Matthieu: TODO : Insérer refs distance de Hamming, Levenstein, String-to-string correction problem (Tichy et al)*

Il conviendra ensuite de déterminer comment établir la valeur seuil à partir de laquelle la distance entre versions est jugée trop importante. Les approches d'évaluation de la qualité du résultat [72] pourront être utilisées pour déterminer un couple ⟨ méthode de calcul de la distance, valeur de distance ⟩ spécifiant les cas pour lesquels les méthodes de résolution de conflits automatiques ne produisent plus un résultat satisfaisant. *Matthieu: TODO : Insérer refs travaux Claudia et Vinh* Le couple obtenu pourra ensuite être confirmé par le biais d'expériences utilisateurs inspirées de [73, 42].

Finalement, il conviendra de proposer un mécanisme de résolution de conflits adapté pour gérer les éventuelles fusions d'une même modification de façon concurrente par un mécanisme automatique et par un mécanisme manuel, ou à défaut un mécanisme de conscience de groupe invitant les utilisateur-rices à effectuer des actions de compensation.

3.2.3 Étude comparative des différents modèles de synchronisation pour CRDTs

Comme évoqué dans l'état de l'art (cf. **??**, page **??**), un nouveau modèle de synchronisation pour CRDT fut proposé récemment [74]. Ce dernier propose une synchronisation des noeuds par le biais de différences d'états. Dans notre étude comparative des différents modèles de synchronisation (cf. **??**, page **??**), nous avons justifié que ce modèle de synchronisation est adapté à l'ensemble des contextes d'utilisation qui étaient jusqu'à lors exclusifs soit au modèle de synchronisation par états, soit par opérations. Dans cette piste de recherche, nous souhaitons approfondir notre étude comparative pour déterminer si le modèle de synchronisation par différences d'états rend obsolètes les modèles de synchronisation précédents.

Pour rappel, ce nouveau modèle de synchronisation se base sur le modèle de synchronisation par états. Il partage les mêmes pré-requis, à savoir la nécessité d'une fonction `merge` associative, commutative et idempotente. Cette dernière doit permettre de la fusion toute paire d'états possible en calculant leur borne supérieure, c.-à-d. leur Least Upper Bound (LUB) [75].

La spécificité de ce nouveau modèle de synchronisation est de calculer pour chaque modification la différence d'état correspondante. Cette différence correspond à un élément irréductible du sup-demi-treillis du CRDT [58], c.-à-d. un état particulier de ce dernier. Cet élément irréductible peut donc être diffusé et intégré par les autres noeuds, toujours à l'aide de la fonction `merge`.

Ce modèle de synchronisation permet alors d'adopter une variété de stratégies de synchronisation, e.g. diffusion des différences de manière atomique, fusion de plusieurs différences puis diffusion du résultat..., et donc de répondre à une grande variété de cas d'utilisation.

Ainsi, un CRDT synchronisé par différences d'états correspond à un CRDT synchronisé par états dont nous avons identifié les éléments irréductibles. La différence entre ces deux modèles de synchronisation semble reposer seulement sur la possibilité d'utiliser ces éléments irréductibles pour propager les modifications, en place et lieu des états complets. Nous conjecturons donc que le modèle de synchronisation par états est rendu obsolète par celui par différences d'états. Il serait intéressant de confirmer cette supposition.

En revanche, l'utilisation du modèle de synchronisation par opérations conduit généralement à une spécification différente du CRDT, les opérations permettant d'encoder plus librement les modifications. Notamment, l'utilisation d'opérations peut mener à des algorithmes d'intégration des modifications différents que ceux de la fonction `merge`. Il convient de comparer ces algorithmes pour déterminer si le modèle de synchronisation par opérations peut présenter un intérêt en termes de surcoût.

Au-delà de ce premier aspect, il convient d'explorer d'autres pistes pouvant induire des avantages et inconvénients pour chacun de ces modèles de synchronisation. À l'issue de cette thèse, nous identifions les pistes suivantes :

(i) La composition de CRDTs, c.-à-d. la capacité de combiner et de mettre en relation plusieurs CRDTs au sein d'un même système, afin d'offrir des fonctionnalités plus complexes. Par exemple, une composition de CRDTs peut se traduire par l'ajout de dépendances entre les modifications des différents CRDTs composés. Le modèle de synchronisation par opérations nous apparaît plus adapté pour cette utilisation, de par le découplage qu'il induit entre les CRDTs et la couche de livraison de messages.

(ii) L'utilisation de CRDTs au sein de systèmes non-sûrs, c.-à-d. pouvant compter un ou plusieurs adversaires byzantins [76]. Dans de tels systèmes, les adversaires byzantins peuvent générer des modifications différentes mais qui sont perçues comme identiques par les mécanismes de résolution de conflits. Cette attaque, nommée *équivoque*, peut provoquer la divergence définitive des copies. [65] propose une solution adaptée aux systèmes P2P à large échelle. Celle-ci se base notamment sur l'utilisation de journaux infalsifiables. *Matthieu: TODO : Ajouter refs* Il convient alors d'étudier si l'utilisation de journaux infalsifiables ne limite pas le potentiel du modèle de synchronisation par différences d'états, e.g. en interdisant la diffusion des modifications par états complets.

Un premier objectif de notre travail serait de proposer des directives sur le modèle de synchronisation à privilégier en fonction du contexte d'utilisation du CRDT.

Ce travail permettrait aussi d'étudier la combinaison des modèles de synchronisation par opérations et par différences d'états au sein d'un même CRDT. Le but serait notamment d'identifier les paramètres conduisant à privilégier un modèle de synchronisation par rapport à l'autre, de façon à permettre aux noeuds de basculer dynamiquement entre les deux.

3.2.4 Approfondissement du patron de conception de Pure Operation-Based CRDTs

Plusieurs approches ont été proposées dans la littérature pour guider la conception de CRDTs :

(i) L'utilisation de la théorie des treillis pour la conception de CRDTs synchronisés par états et par différences d'états [22, 58].

(ii) L'utilisation d'un journal partiellement ordonné des opérations, nommé PO-Log, pour la conception de CRDTs synchronisés par opérations [57].

Cependant, ce framework proposé par [57] souffre de plusieurs limitations. Nous souhaitons donc proposer un nouveau framework pour la conception de CRDTs synchronisés par opérations, en nous basant sur ce dernier.

Le framework proposé dans [57] possède plusieurs objectifs :

(i) Proposer une approche partiellement générique pour définir un CRDT synchronisé par opérations.

(ii) Factoriser les métadonnées utilisées par le CRDT pour le mécanisme de résolution de conflits, notamment pour identifier les éléments, et celles utilisées par la couche livraison, notamment pour identifier les opérations.

(iii) Inclure des mécanismes de GC de ces métadonnées pour réduire la taille de l'état.

Pour cela, les auteurs se limitent aux CRDTs purs synchronisés par opérations, c.-à-d. les CRDTs dont les modifications enrichies de leurs arguments et d'une estampille fournie par la couche de livraison des messages sont commutatives. Pour ces CRDTs, les auteurs proposent un framework générique permettant leur spécification sous la forme d'un PO-Log. Les auteurs associent le PO-Log à une couche de livraison Reliable Causal Broadcast (RCB) des opérations.

Les auteurs définissent ensuite le concept de stabilité causale. Ce concept leur permet de retirer les métadonnées de causalité des opérations du PO-Log lorsque celles-ci sont déterminées comme étant causalement stables.

Finalement, les auteurs définissent un ensemble de relations, spécifiques à chaque CRDT, qui permettent d'exprimer la *redondance causale*. La redondance causale permet de spécifier quand retirer une opération du PO-Log, car rendue obsolète par une autre opération.

Comme évoqué précédemment, cette approche souffre toutefois de plusieurs limites. Tout d'abord, elle repose sur l'utilisation d'une couche de livraison RCB. Cette couche satisfait le modèle de livraison causale. Mais pour rappel, ce modèle induit l'ajout de données de causalité précises à chaque opération, sous la forme d'un vecteur de version ou d'une barrière causale. Nous jugeons ce modèle trop coûteux pour les systèmes P2P dynamiques à large échelle sujets au churn.

En plus du coût induit en termes de métadonnées et de bande-passante, le modèle de livraison causale peut aussi introduire un délai superflu dans la livraison des opérations. En effet, ce modèle impose que tous les messages précédant un nouveau message d'après la

relation *happens-before* soient eux-mêmes livrés avant de livrer ce dernier. Il en résulte que des opérations peuvent être mises en attente par la couche livraison, e.g. suite à la perte d'une de leurs dépendances d'après la relation *happens-before*, alors que leurs dépendances réelles ont déjà été livrées et que les opérations sont de fait intégrables en l'état. Plusieurs travaux [77, 78] ont noté ce problème. Pour y répondre et ainsi améliorer la réactivité du framework Pure Operation-Based, ils proposent d'exposer les opérations mises en attente par la couche livraison au CRDT. Bien que fonctionnelle, cette approche induit toujours le coût d'une couche de livraison respectant le modèle de livraison causale et nous fait considérer la raison de ce coût, le modèle de livraison n'étant dès lors plus respecté.

Ensuite, ce framework impose que la modification **prepare** ne puisse pas inspecter l'état courant du noeud. Cette contrainte est compatible avec les CRDTs pour les types de données simples qui sont considérés dans [57], e.g. le Compteur ou l'Ensemble. Elle empêche cependant l'expression de CRDTs pour des types de données plus complexes, e.g. la Séquence ou le Graphe. *Matthieu: TODO : À confirmer pour le graphe* Nous jugeons dommageable qu'un framework pour la conception de CRDTs limite de la sorte son champ d'application.

Finalement, les auteurs ne considèrent que des types de données avec des modifications à granularité fixe. Ainsi, ils définissent la notion de redondance causale en se limitant à ce type de modifications. Par exemple, ils définissent que la suppression d'un élément d'un ensemble rend obsolète les ajouts précédents de cet élément. Cependant, dans le cadre d'autres types de données, e.g. la Séquence, une modification peut concerner un ensemble d'éléments de taille variable. Une opération peut donc être rendue obsolète non pas par une opération, mais par un ensemble d'opérations. Par exemple, les suppressions d'éléments formant une sous-chaîne rendent obsolète l'insertion de cette sous-chaîne. Ainsi, la notion de redondance causale est incomplète et souffre de l'absence d'une notion d'obsolescence partielle d'une opération.

Pour répondre aux différents problèmes soulevés, nous souhaitons proposer un nouveau framework en nous basant sur [57]. Nos objectifs sont les suivants :

(i) Proposer un framework mettant en lumière la présence et le rôle de deux modèles de livraison :
  (i) Le modèle de livraison minimal requis par le CRDT pour assurer la convergence forte à terme [22].
  (ii) Le modèle de livraison employé par le système qui utilise le CRDT. Ce second modèle de livraison est une stratégie permettant au système de respecter un modèle de cohérence donné et régissant les règles de compaction de l'état. Il doit être égal ou plus contraint que modèle de livraison minimal du CRDT et peut être amené à évoluer en fonction de l'état du système et de ses besoins. Par exemple, un système pourrait par défaut utiliser le modèle de livraison causale pour assurer le modèle de cohérence causal. Puis, lorsque le nombre de noeuds atteint un seuil donné et que le coût de la livraison causale devient trop élevé, le système pourrait passer au modèle de livraison FIFO pour assurer le modèle de cohérence PRAM afin de réduire les coûts en bande-passante.

(ii) Étendre la notion de redondance causale pour prendre en compte la redondance

partielle des opérations. De plus, nous souhaitons rendre cette notion accessible à la couche de livraison, pour détecter au plus tôt les opérations désormais obsolètes et prévenir leur diffusion.

(iii) Identifier et classifier les mécanismes de résolution de conflits, pour déterminer lesquels sont indépendants de l'état courant pour la génération des opérations et lesquels nécessitent d'inspecter l'état courant dans **prepare**.

Annexe A

Entrelacement d'insertions concurrentes dans Treedoc

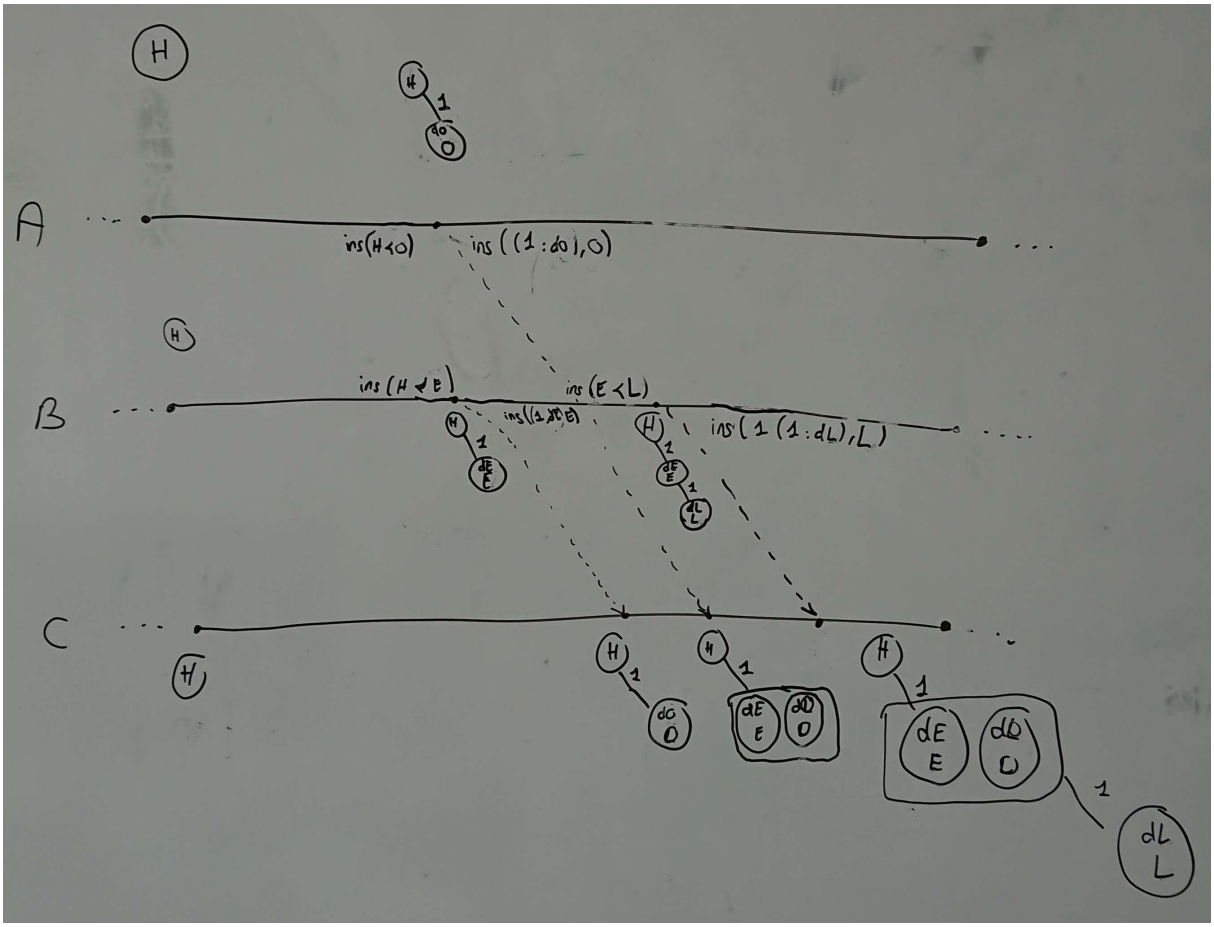


FIGURE A.1 – Modifications concurrentes d'une séquence Treedoc résultant en un entrelacement

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 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 C

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] Jonathan A Obar et Steven S Wildman. « Social Media Definition and the Governance Challenge - An Introduction to the Special Issue ». In : *Obar, JA and Wildman, S.(2015). Social media definition and the governance challenge : An introduction to the special issue. Telecommunications policy* 39.9 (2015), p. 745–750.

[2] Statista. *Biggest social media platforms 2022*. Last Accessed : 2022-10-06. URL : https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/.

[3] Wikimedia. *Wikimedia Statistics - English Wikipedia*. Last Accessed : 2022-10-06. URL : https://stats.wikimedia.org/#/en.wikipedia.org.

[4] David Meek. « YouTube and Social Movements : A Phenomenological Analysis of Participation, Events and Cyberplace ». In : *Antipode* 44.4 (2012), p. 1429–1448. DOI : https://doi.org/10.1111/j.1467-8330.2011.00942.x. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8330.2011.00942.x. URL : https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8330.2011.00942.x.

[5] Yannis Theocharis. « The wealth of (occupation) networks ? Communication patterns and information distribution in a Twitter protest network ». In : *Journal of Information Technology & Politics* 10.1 (2013), p. 35–56.

[6] José Van Dijck et Thomas Poell. « Understanding social media logic ». In : *Media and communication* 1.1 (2013), p. 2–14.

[7] Jim Giles. « Special Report Internet encyclopaedias go head to head ». In : *nature* 438.15 (2005), p. 900–901.

[8] Lada A Adamic, Jun Zhang, Eytan Bakshy et Mark S Ackerman. « Knowledge sharing and yahoo answers : everyone knows something ». In : *Proceedings of the 17th international conference on World Wide Web*. 2008, p. 665–674.

[9] Paul Baran. « On distributed communications networks ». In : *IEEE transactions on Communications Systems* 12.1 (1964), p. 1–9.

[10] Safiya Umoja Noble. *Algorithms of Oppression : How Search Engines Reinforce Racism*. NYU Press, 2018. ISBN : 9781479849949.

[11] Amnesty International. *#Toxictwitter : Violence and abuse against women online*. Last Accessed : 2022-10-07. URL : https://www.amnesty.org/en/documents/act30/8070/2018/en/.

[12] Wall Street Journal. *Facebook Tried to Make Its Platform a Healthier Place. It Got Angrier Instead.* Last Accessed : 2022-10-07. URL : https://t.co/P6JohMdhQE.

[13] Wall Street Journal. *Facebook Knows Instagram Is Toxic for Teen Girls, Company Documents Show.* Last Accessed : 2022-10-07. URL : https://t.co/JAvzKFc61q.

[14] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg et Mark McGranaghan. « 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] Daniel Abadi. « Consistency Tradeoffs in Modern Distributed Database System Design : CAP is Only Part of the Story ». In : *Computer* 45.2 (2012), p. 37–42. DOI : 10.1109/MC.2012.33.

[16] Yasushi Saito et Marc Shapiro. « Optimistic Replication ». In : *ACM Comput. Surv.* 37.1 (mar. 2005), p. 42–81. ISSN : 0360-0300. DOI : 10.1145/1057977.1057980. URL : https://doi.org/10.1145/1057977.1057980.

[17] Douglas B Terry, Marvin M Theimer, Karin Petersen, Alan J Demers, Mike J Spreitzer et Carl H Hauser. « Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System ». In : *SIGOPS Oper. Syst. Rev.* 29.5 (déc. 1995), p. 172–182. ISSN : 0163-5980. DOI : 10.1145/224057.224070. URL : https://doi.org/10.1145/224057.224070.

[18] Daniel Stutzbach et Reza Rejaie. « Understanding Churn in Peer-to-Peer Networks ». In : *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement*. IMC '06. Rio de Janeriro, Brazil : Association for Computing Machinery, 2006, p. 189–202. ISBN : 1595935614. DOI : 10.1145/1177080.1177105. URL : https://doi.org/10.1145/1177080.1177105.

[19] Leslie Lamport. « The part-time parliament ». In : *Concurrency : the Works of Leslie Lamport*. 2019, p. 277–317.

[20] Diego Ongaro et John Ousterhout. « In search of an understandable consensus algorithm ». In : *2014 USENIX Annual Technical Conference (Usenix ATC 14)*. 2014, p. 305–319.

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

[22] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero et Marek Zawirski. « 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.

[23] 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.

[24] 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.

[25] Google. *Google Docs*. Last Accessed : 2022-10-07. URL : https://docs.google.com/.

[26] Cody Odgen. *Google Graveyard*. Last Accessed : 2022-10-11. URL : https://killedbygoogle.com/.

[27] 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.

[28] 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.

[29] 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.

[30] Matthieu Nicolas, Gerald Oster et Olivier Perrin. « Efficient Renaming in Sequence CRDTs ». In : *IEEE Transactions on Parallel and Distributed Systems* 33.12 (déc. 2022), p. 3870–3885. DOI : 10.1109/TPDS.2022.3172570. URL : https://hal.inria.fr/hal-03772633.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] Sylvie NOËL et Jean-Marc ROBERT. « Empirical study on collaborative writing : What do co-authors do, use, and like ? » In : *Computer Supported Cooperative Work (CSCW)* 13.1 (2004), p. 63–89.

[40] Clarence A. ELLIS et Simon J. GIBBS. « Concurrency Control in Groupware Systems ». In : *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD '89. Portland, Oregon, USA : Association for Computing Machinery, 1989, p. 399–407. ISBN : 0897913175. DOI : 10.1145/67544.66963. URL : https://doi.org/10.1145/67544.66963.

[41] ETHERPAD. *Etherpad*. Last Accessed : 2022-10-07. URL : https://etherpad.org/.

[42] Claudia-Lavinia IGNAT, Gérald OSTER, Olivia FOX, François CHAROY et Valerie SHALIN. « How Do User Groups Cope with Delay in Real-Time Collaborative Note Taking ». In : *European Conference on Computer Supported Cooperative Work 2015*. Sous la dir. de Nina BOULUS-RODJE, Gunnar ELLINGSEN, Tone BRATTETEIG, Margunn AANESTAD et Pernille BJORN. Proceedings of the 14th European Conference on Computer Supported Cooperative Work. Oslo, Norway : Springer International

Publishing, sept. 2015, p. 223–242. DOI : 10.1007/978-3-319-20499-4_12. URL : https://hal.inria.fr/hal-01238831.

[43] Quang-Vinh DANG et Claudia-Lavinia IGNAT. « Performance of real-time collaborative editors at large scale : User perspective ». In : *Internet of People Workshop, 2016 IFIP Networking Conference*. Proceedings of 2016 IFIP Networking Conference, Networking 2016 and Workshops. Vienna, Austria, mai 2016, p. 548–553. DOI : 10.1109/IFIPNetworking.2016.7497258. URL : https://hal.inria.fr/hal-01351229.

[44] Barton GELLMAN et Laura POITRAS. *U.S., British intelligence mining data from nine U.S. Internet companies in broad secret program*. Last Accessed : 2022-10-07. URL : https://www.washingtonpost.com/investigations/us-intelligence-mining-data-from-nine-us-internet-companies-in-broad-secret-program/2013/06/06/3a0c0da8-cebf-11e2-8845-d970ccb04497_story.html.

[45] Glen GREENWALD et Ewen MACASKILL. *NSA Prism program taps in to user data of Apple, Google and others*. Last Accessed : 2022-10-07. URL : https://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[46] Mehdi AHMED-NACER, Claudia-Lavinia IGNAT, Gérald OSTER, Hyun-Gul ROH et Pascal URSO. « 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.

[47] 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.

[48] Jim PICK. *PeerPad*. Last Accessed : 2022-10-07. URL : https://peerpad.net/.

[49] Jim PICK. *Graf, Nikolaus*. Last Accessed : 2022-10-07. URL : https://www.serenity.re/en/notes.

[50] 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.

[51] Paulo Sérgio ALMEIDA, Carlos BAQUERO, Ricardo GONÇALVES, Nuno PREGUIÇA et Victor FONTE. « Scalable and Accurate Causality Tracking for Eventually Consistent Stores ». In : *Distributed Applications and Interoperable Systems*. Sous la dir. de Kostas MAGOUTIS et Peter PIETZUCH. Berlin, Heidelberg : Springer Berlin Heidelberg, 2014, p. 67–81. ISBN : 978-3-662-43352-2.

[52] Madhavan MUKUND, Gautham SHENOY et SP SURESH. « Optimized or-sets without ordering constraints ». In : *International Conference on Distributed Computing and Networking*. Springer. 2014, p. 227–241.

[53] D. S. Parker, G. J. Popek, G. Rudisin, A. Stoughton, B. J. Walker, E. Walton, J. M. Chow, D. Edwards, S. Kiser et C. Kline. « 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.

[54] Nuno M. Preguiça. « Conflict-free Replicated Data Types : An Overview ». In : *CoRR* abs/1806.10254 (2018). arXiv : 1806.10254. URL : http://arxiv.org/abs/1806.10254.

[55] Weihai Yu et Sigbjørn Rostad. « A Low-Cost Set CRDT Based on Causal Lengths ». In : *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*. New York, NY, USA : Association for Computing Machinery, 2020. ISBN : 9781450375245. URL : https://doi.org/10.1145/3380787.3393678.

[56] 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.

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

[58] Vitor Enes, Paulo Sérgio Almeida, Carlos Baquero et João Leitão. « Efficient Synchronization of State-Based CRDTs ». In : *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019, p. 148–159. DOI : 10.1109/ICDE.2019.00022.

[59] Gérald Oster, Pascal Urso, Pascal Molli et Abdessamad Imine. « 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.

[60] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim et Joonwon Lee. « 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.

[61] Nuno Preguica, Joan Manuel Marques, Marc Shapiro et Mihai Letia. « 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.

[62] 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.

[63] Brice Nédelec, Pascal Molli, Achour Mostéfaoui et Emmanuel Desmontils. « LSEQ : an adaptive structure for sequences in distributed collaborative editing ». In : *Proceedings of the 2013 ACM Symposium on Document Engineering*. DocEng 2013. Sept. 2013, p. 37–46. DOI : 10.1145/2494266.2494278.

[64] Brice Nédelec, Pascal Molli et Achour Mostéfaoui. « A scalable sequence encoding for collaborative editing ». In : *Concurrency and Computation : Practice and Experience* (), e4108. DOI : 10.1002/cpe.4108. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4108. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4108.

[65] Victorien Elvinger, Gérald Oster et Francois Charoy. « Prunable Authenticated Log and Authenticable Snapshot in Distributed Collaborative Systems ». In : *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. 2018, p. 156–165. DOI : 10.1109/CIC.2018.00031.

[66] OpenRelay. *OpenRelay*. Last Accessed : 2022-10-07. URL : https://openrelay.xyz/.

[67] Protocol Labs. *IPFS*. Last Accessed : 2022-10-07. URL : https://ipfs.io/.

[68] Friedemann Mattern et al. *Virtual time and global states of distributed systems*. Univ., Department of Computer Science, 1988.

[69] Colin Fidge. « Logical Time in Distributed Computing Systems ». In : *Computer* 24.8 (août 1991), p. 28–33. ISSN : 0018-9162. DOI : 10.1109/2.84874. URL : https://doi.org/10.1109/2.84874.

[70] Ravi Prakash, Michel Raynal et Mukesh Singhal. « An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments ». In : *Journal of Parallel and Distributed Computing* 41.2 (1997), p. 190–204. ISSN : 0743-7315. DOI : https://doi.org/10.1006/jpdc.1996.1300. URL : https://www.sciencedirect.com/science/article/pii/S0743731596913003.

[71] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan et Alastair R. Beresford. « 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.

[72] Quang Vinh Dang et Claudia-Lavinia Ignat. « Quality Assessment of Wikipedia Articles : A Deep Learning Approach by Quang Vinh Dang and Claudia-Lavinia Ignat with Martin Vesely as Coordinator ». In : *SIGWEB Newsl.* Autumn (nov. 2016). ISSN : 1931-1745. DOI : 10.1145/2996442.2996447. URL : https://doi.org/10.1145/2996442.2996447.

[73] Claudia-Lavinia IGNAT, Gérald OSTER, Meagan NEWMAN, Valerie SHALIN et François CHAROY. « Studying the Effect of Delay on Group Performance in Collaborative Editing ». In : *Proceedings of 11th International Conference on Cooperative Design, Visualization, and Engineering, CDVE 2014, Springer 2014 Lecture Notes in Computer Science*. Proceedings of 11th International Conference on Cooperative Design, Visualization, and Engineering, CDVE 2014. Seattle, WA, United States, sept. 2014, p. 191–198. DOI : 10.1007/978-3-319-10831-5_29. URL : https://hal.archives-ouvertes.fr/hal-01088815.

[74] 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.

[75] Nuno M. PREGUIÇA, Carlos BAQUERO et Marc SHAPIRO. « Conflict-free Replicated Data Types (CRDTs) ». In : *CoRR* abs/1805.06358 (2018). arXiv : 1805.06358. URL : http://arxiv.org/abs/1805.06358.

[76] Leslie LAMPORT, Robert SHOSTAK et Marshall PEASE. « The Byzantine Generals Problem ». In : *Concurrency : The Works of Leslie Lamport*. New York, NY, USA : Association for Computing Machinery, 2019, p. 203–226. ISBN : 9781450372701. URL : https://doi.org/10.1145/3335772.3335936.

[77] Jim BAUWENS et Elisa Gonzalez BOIX. « Flec : A Versatile Programming Framework for Eventually Consistent Systems ». 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 : 10.1145/3380787.3393685. URL : https://doi.org/10.1145/3380787.3393685.

[78] Jim BAUWENS et Elisa Gonzalez BOIX. « Improving the Reactivity of Pure Operation-Based 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 : 10.1145/3447865.3457968. URL : https://doi.org/10.1145/3447865.3457968.

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