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

Introduction

1 Contexte

— Systèmes collaboratifs (wikis, plateformes de contenu, réseaux sociaux) et leurs bienfaits (qualité de l'info, vitesse de l'info (exemple de crise ?), diffusion de la parole). Démocratisation (sic) de ces systèmes au cours de la dernière décennie.

— En raison du volume de données et de requêtes, adoptent architecture décentralisée. Permet ainsi de garantir disponibilité, tolérance aux pannes et capacité de passage à l'échelle.

— Mais échoue à adresser problèmes non-techniques : confidentialité, souveraineté, protection contre censure, dépendance et nécessité de confiance envers autorité centrale.

— À l'heure où les entreprises derrière ces systèmes font preuve d'ingérence et d'intérêts contraires à ceux de leurs utilisateur-rices (Cambridge Analytica, Prism, non-modération/mise en avant de contenus racistes [1] [2] [3], invisibilisation de contenus féministes, dissolution du comité d'éthique de Google [4], inégalité d'accès à la méta-machine affectante [5] [6] [7]), parait fondamental de proposer les moyens technologiques accessibles pour concevoir et déployer des alternatives.

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

— Systèmes pair-à-pair sont une direction intéressante pour répondre à ces problématiques, de part leur absence d'autorité centrale, la distribution des tâches et leur conception mettant le pair au centre. Mais posent de nouvelles problématiques de recherche.

— Ces systèmes ne disposent d'aucun contrôle sur les noeuds qui les composent. Le nombre de noeuds peut donc croître de manière non-bornée et atteindre des centaines de milliers de noeuds. La complexité des algorithmes de ces systèmes ne doit donc pas dépendre de ce paramètre, ou alors de manière logarithmique.

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

— De plus, ces noeuds n'offrent aucune garantie sur leur stabilité. Ils peuvent donc rejoindre et participer au système de manière éphèmère. S'agit du phénomène connu sous le nom de churn. Les algorithmes de ces systèmes ne peuvent donc pas reposer sur des mécanismes nécessitant une coordination synchrone d'une proportion des noeuds.

— Finalement, ces noeuds n'offrent aucune garanties sur leur fiabilité et intentions. Les noeuds peuvent se comporter de manière byzantine. Pour assurer la confidentialité, l'absence de confiance requise et le bon fonctionnement du système, ce dernier doit être conçu pour résister aux comportements byzantins de ses acteurs.

— Ainsi, il est nécessaire de faire progresser les technologies existantes pour les rendre compatible avec ce nouveau modèle de système. Dans le cadre de cette thèse, nous nous intéressons aux mécanismes de réplication de données dans les systèmes collaboratifs pair-à-pair temps réel.

2 Questions de recherche et contributions

2.1 Ré-identification sans coordination synchrone pour CRDTs pour le type Séquence

— Les systèmes collaboratifs permettent à des utilisateur-rices de manipuler et d'éditer un contenu partagé. Ces systèmes peuvent adopter le paradigme de la réplication optimiste [1] pour favoriser leur disponibilité, latence et tolérance aux pannes. Ce paradigme autorise les noeuds possédant une copie de la donnée partagée de la consulter et de la modifier sans se coordonner préalablement avec les autres noeuds. Leur copies peuvent alors diverger momentanément. Un mécanisme de synchronisation permet ensuite à chaque noeud de récupérer l'ensemble des modifications et de les intégrer de façon à converger, c.-à-d. obtenir de nouveau des états équivalents. Cependant, le paradigme de la réplication optimiste permet la génération en concurrence de modifications provoquant un conflit, e.g. l'ajout et la suppression d'un même élément dans un Ensemble. Un mécanisme de résolution de conflits est alors nécessaire pour assurer la convergence à terme des noeuds [2].

— Les CRDTs [3, 4] sont des types de données répliqués. Ils sont conçus pour être répliqués par les noeuds d'un système et pour permettre à ces derniers de modifier les données partagées sans aucune coordination. Dans ce but, ils incluent des mécanismes de résolution de conflits automatiques directement au sein leur spécification. Ces mécanismes leur permettent de résoudre le problème évoqué précédemment. Cependant, ces mécanimes induisent un surcoût, aussi bien en termes de métadonnées, calculs et bande-passante. Notamment, certains CRDTs comme ceux pour le type Séquence souffrent d'une croissance monotone de leur surcoût. Ce surcoût s'avère handicapant dans le contexte des collaborations à large échelle.

— Dans le contexte des CRDTs pour le type Séquence, le surcoût du type de données répliquées provient de la croissance des métadonnées. Notamment, ces métadonnées correspondent à des identifiants associés aux éléments de la Séquence par les CRDTs.

Ces identifiants sont ensuite utilisées par leur mécanisme de résolution de conflits automatique pour .

— Plusieurs approches ont été proposées pour réduire le coût induit par ces identifiants. Notamment, [5, 6] proposent un mécanisme de ré-assignation des identifiants pour réduire leur taille. Ce mécanisme provoque cependant des conflits avec les modifications concurrentes de la Séquence, c.-à-d. l'insertion ou la suppression. Pour résoudre ces nouveaux conflits, les auteurs proposent un mécanisme de transformation des modifications concurrentes par rapport au mécanisme de ré-assignation.

— Cependant, des exécutions concurrentes du mécanisme de ré-assignation des identifiants provoquent elles aussi des conflits. Pour éviter ces dernier, les auteurs choisissent de subordonner l'exécution du mécanisme de ré-assignation à un protocole de consensus. Ainsi, le mécanisme de ré-assignation ne peut être déclenché en concurrence par les noeuds du systèmes.

— Cependant, reposer sur un protocole de consensus est une contrainte forte dans un système distribué. Nous la jugeons même prohibitive pour 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. compatible avec les systèmes P2P à large échelle sujets au churn ?*

— Nous répondons à cette problématique en proposant RenamableLogootSplit, un nouveau CRDT pour le type Séquence. Ce CRDT intègre un mécanisme de renommage directement au sein de sa spécification, ainsi qu'un mécanisme de résolution de conflits automatique pour résoudre le conflit provoqué par des déclenchements concurrents de ce dernier. Ainsi, nous proposons un mécanisme de renommage permettant de réduire le surcoût du CRDT pour Séquence et qui est utilisable par les noeuds sans aucune coordination synchrone entre eux.

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 [7]. 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 jouent donc un rôle central 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 de ces 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 des serveurs, les utilisateur-rices de ces systèmes dépendent alors de ces autorités centrales et de leurs intérêts. Par exemple, les autorités centrales ayant le pouvoir de vie et de mort sur leurs systèmes collaboratifs, elles menacent la pérénnité de ces systèmes.

— Pour répondre à ces problématiques, c.-à-d. confidentialité et souveraineté des données, dépendance envers des tiers, pérénnité des systèmes, un nouveau paradigme d'applications proposent de concevoir des Local-First Softwares (LFS), c.-à-d. des applications mettant l'utilisateur-rice et son appareil au coeur du système. Dans ce cadre d'applications, les serveurs sont relégués à un rôle de support à la collaboration.

— Dans le cadre de cette mouvance, notre équipe de recherche étudie notamment la conception de systèmes P2P sans autorités centrales. 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 connexion 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 Multi User Text Editor (MUTE) [8]. Il s'agit d'un éditeur de texte web collaboratif P2P temps réel chiffré de bout en bout. Ce projet permet de présenter les travaux de recherche de l'équipe, c.-à-d. mécanismes de résolutions de conflits automatiques pour le type Séquence [9, 10, 11] et mécanisme d'authentification des pairs sans autorités centrales [12, 13]. Il permet aussi d'offrir un tour d'horizon des nombreux travaux de recherche nécessaires à la conception de tels systèmes, c.-à-d. mécanismes de conscience de groupe *Matthieu: TODO : Trouver et ajouter références*, protocoles d'appartenance aux groupes [14, 15], topologies réseaux P2P [16] et protocoles d'établissement de clés de chiffrement de groupe [17]. À notre connaissance, il s'agit du Proof of Concept (PoC) le plus complet d'applications LFS P2P sans autorités centrales. *Matthieu: TODO : Vérifier du côté des applis de IPFS*

3 Plan du manuscrit

— Ce manuscrit de thèse est organisé de la manière suivante :

— Dans le chapitre 1, nous introduisons le modèle du système que nous considérons, c.-à-d. les systèmes P2P à large échelle sujet au churn. 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 sujet 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 **??**, 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 2 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.

4 Publications

Chapitre 1

État de l'art

Sommaire

1.1 Modèle du système

Le système que nous considérons est un système Pair-à-Pair (P2P) à large échelle. Il est composé d'un ensemble de noeuds dynamique. En d'autres termes, un noeud peut rejoindre ou quitter le système à tout moment.

À un instant donné, un noeud est soit connecté, soit déconnecté. Nous considérons possible qu'un noeud se déconnecte de manière définitive, sans indication au préalable. Ainsi, du point de vue des autres noeuds du système, il est impossible de déterminer le statut d'un noeud déconnecté. Ce dernier peut être déconnecté de manière temporaire ou définitive. Toutefois, nous assimilons les noeuds déconnectés de manière définitive à des noeuds ayant quittés le système, ceux-ci ne participant plus au système.

Dans ce système, nous considérons comme confondus les noeuds et clients. Un noeud correspond alors à un appareil d'un-e utilisateur-rice du système. Un-e même utilisateur-rice peut prendre part au système au travers de différents appareils, nous considérons alors chaque appareil comme un noeud distinct.

Le système consiste en une application permettant de répliquer une donnée. Chaque noeud du système possède en local une copie de la donnée. Les noeuds peuvent consulter et éditer leur copie locale à tout moment, sans se coordonner entre eux. Les modifications sont appliquées à la copie locale immédiatement et de manière atomique. Les modifications sont ensuite transmises aux autres noeuds de manière asynchrone par le biais de messages, afin qu'ils puissent à leur tour intégrer les modifications à leur copie. L'application garantit la convergence à terme des copies.

Définition 1 (Convergence à terme)**.** La convergence à terme est une propriété de sûreté indiquant que l'ensemble des noeuds du système ayant intégrés le même ensemble de

modifications obtiendront des états équivalents [8].

Les noeuds communiquent entre eux par l'intermédiaire d'un réseau non-fiable. Les messages envoyés peuvent être perdus, ré-ordonnés et/ou dupliqués. Le réseau est aussi sujet à des partitions, qui séparent les noeuds en des sous-groupes disjoints. Aussi, nous considérons que les noeuds peuvent initier de leur propre chef des partitions réseau : des groupes de noeuds peuvent décider de travailler de manière isolée pendant une certaine durée, avant de se reconnecter au réseau.

Pour compenser les limitations du réseau, les noeuds reposent sur une couche de livraison de messages. Cette couche permet de garantir un modèle de livraison donné des messages à l'application. En fonction des garanties du modèle de livraison sélectionné, cette couche peut ré-ordonner les messages reçus avant de les livrer à l'application, dédupliquer les messages, et détecter et ré-échanger les messages perdus. Nous considérons a minima que la couche de livraison garantit la livraison à terme des messages.

Définition 2 (Livraison à terme)**.** La livraison à terme est un modèle de livraison garantissant que l'ensemble des messages du système seront livrés à l'ensemble des noeuds du système à terme.

Finalement, nous supposons que les noeuds du système sont honnêtes. Les noeuds ne peuvent dévier du protocole de la couche de livraison des messages ou de l'application. Les noeuds peuvent cependant rencontrer des défaillances. Nous considérons que les noeuds disposent d'une mémoire durable et fiable. Ainsi, nous considérons que les noeuds peuvent restaurer le dernier état valide, c.-à-d. pas en cours de modification, qu'il possédait juste avant la défaillance.

1.2 Types de données répliquées sans conflits

Afin d'offrir une haute disponibilité à leurs clients et afin d'accroître leur tolérance aux pannes [18], les systèmes distribués peuvent adopter le paradigme de la réplication optimiste [1]. Ce paradigme consiste à ce que chaque noeud composant le système possède une copie de la donnée répliquée. Chaque noeud possède le droit de la consulter et de la modifier, sans coordination préalable avec les autres noeuds. Les noeuds peuvent alors temporairement diverger, c.-à-d. posséder des états différents. Un mécanisme de synchronisation leur permet ensuite de partager leurs modifications respectives et d'obtenir de nouveau des états équivalent, c.-à-d. de converger à terme [2].

Pour permettre aux noeuds de converger, les protocoles de réplication optimiste ordonnent généralement les évènements se produisant dans le système distribué. Pour les ordonner, la littérature repose généralement sur la relation de causalité entre les évènements, qui est définie par la relation *happens-before* [19]. Nous l'adaptons ci-dessous à notre contexte, en ne considérant que les modifications [9] effectuées et celles intégrées :

8. Nous considérons comme équivalents deux états pour lesquels chaque observateur du type de données renvoie un même résultat, c.-à-d. les deux états sont indifférenciables du point de vue des utilisateurrices du système.

9. Nous utilisons le terme *modifications* pour désigner les *opérations de modifications* des types abstraits de données afin d'éviter une confusion avec le terme *opération* introduit ultérieurement.

Définition 3 (Relation *happens-before*)**.** La relation *happens-before* indique qu'une modification m_1 a eu lieu avant une modification m_2, notée $m_1 \to m_2$, si et seulement si une des conditions suivantes est satisfaite :

(i) m_1 a été effectuée avant m_2 sur le même noeud.

(ii) m_1 a été intégrée par le noeud auteur de m_2 avant qu'il n'effectue m_2.

(iii) Il existe une modification m telle que $m_1 \to m \wedge m \to m_2$.

Dans le cadre d'un système distribué, nous notons que la relation *happens-before* ne permet pas d'établir un ordre total entre les modifications. En effet, deux modifications m_1 et m_2 peuvent être effectuées en parallèle par deux noeuds différents, sans avoir connaissance de la modification de leur pair respectif. De telles modifications sont alors dites *concurrentes* :

Définition 4 (Concurrence)**.** Deux modifications m_1 et m_2 sont concurrentes, noté $m_1 \parallel m_2$, si et seulement si $m_1 \nrightarrow m_2 \wedge m_1 \nrightarrow m_2$.

Lorsque les modifications possibles sur un type de données sont commutatives, l'intégration des modifications effectuées par les autres noeuds, même concurrentes, ne nécessite aucun mécanisme particulier. Cependant, les modifications permises par un type de données ne sont généralement pas commutatives car de sémantiques contraires, e.g. l'ajout et la suppression d'un élément dans une Collection. Ainsi, une exécution distribuée peut mener à la génération de modifications concurrentes non commutatives. Nous parlons alors de conflits.

Avant d'illustrer notre propos avec un exemple, nous introduisons la spécification algébrique du type Ensemble dans la Figure 1.1 sur laquelle nous nous basons.

payload
$S \in Set\langle E \rangle$

constructor

emp	:		$\longrightarrow S$

mutators

add	:	$S \times E$	$\longrightarrow S$
rmv	:	$S \times E$	$\longrightarrow S$

queries

len	:	S	$\longrightarrow \mathbb{N}$
rd	:	S	$\longrightarrow S$

FIGURE 1.1 – Spécification algébrique du type abstrait usuel Ensemble

Un Ensemble est une collection dynamique non-ordonnée d'éléments de type E. Cette spécification définit que ce type dispose d'un constructeur, *emp*, permettant de générer un ensemble vide.

La spécification définit deux modifications sur l'ensemble :

(i) $add(s, e)$, qui permet d'ajouter un élément donné e à un ensemble s. Cette modification renvoie un nouvel ensemble construit de la manière suivante :

$$add(s, e) = s \cup \{e\}$$

(ii) $rmv(s, e)$, qui permet de retirer un élément donné e d'un ensemble s. Cette modification renvoie un nouvel ensemble construit de la manière suivante :

$$rmv(s, e) = s \setminus \{e\}$$

Elle définit aussi deux observateurs :

(i) $len(s)$, qui permet de récupérer le nombre d'éléments présents dans un ensemble s.

(ii) $rd(s)$, qui permet de consulter l'état d'ensemble s. Dans le cadre de nos exemples, nous considérons qu'une consultation de l'état est effectuée de manière implicite à l'aide de *rd* après chaque modification.

Dans le cadre de ce manuscrit, nous travaillons sur des ensembles de caractères. Cette restriction du domaine se fait sans perte en généralité. En se basant sur cette spécification, nous présentons dans la Figure 1.2 un scénario où des noeuds effectuent en concurrence des modifications provoquant un conflit.

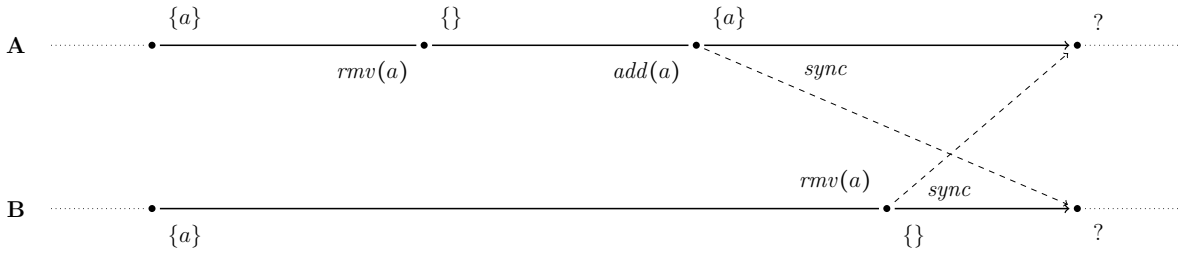

FIGURE 1.2 – Modifications concurrentes d'un Ensemble répliqué provoquant un conflit suite à l'ajout et la suppression d'un même élément

Dans cet exemple, deux noeuds A et B répliquent et partagent une même structure de données de type Ensemble. Les deux noeuds possèdent le même état initial : $\{a\}$. Le noeud A retire l'élément a de l'ensemble, en procédant à la modification $rmv(a)$. Puis, le noeud A ré-ajoute l'élément a dans l'ensemble via la modification $add(a)$. En concurrence, le noeud B retire lui aussi l'élément a de l'ensemble. Les deux noeuds se synchronisent ensuite.

À l'issue de ce scénario, l'état à produire n'est pas trivial : le noeud A a exprimé son intention d'ajouter l'élément a à l'ensemble, tandis que le noeud B a exprimé son intention contraire de retirer l'élément a de ce même ensemble. Ainsi, les états $\{a\}$ et $\{\}$ semblent tous les deux corrects et légitimes dans cette situation. Il est néanmoins primordial que les noeuds choisissent et convergent vers un même état pour leur permettre de poursuivre leur collaboration. Pour ce faire, il est nécessaire de mettre en place un mécanisme de résolution de conflits, potentiellement automatique.

Les Conflict-free Replicated Data Types (CRDTs) [4, 20, 21] répondent à ce besoin.

Définition 5 (Conflict-free Replicated Data Type)**.** Les CRDTs sont de nouvelles spécifications des types de données existants, e.g. l'Ensemble ou la Séquence. Ces nouvelles spécifications sont conçues pour être utilisées dans des systèmes distribués adoptant la réplication optimiste. Ainsi, elles offrent les deux propriétés suivantes :

(i) Les CRDTs peuvent être modifiés sans coordination avec les autres noeuds.

(ii) Les CRDTs garantissent la *convergence forte* [4].

Définition 6 (Convergence forte)**.** La convergence forte est une propriété de sûreté indiquant que l'ensemble des noeuds d'un système ayant intégrés le même ensemble de modifications obtiendront des états équivalents, sans échange de message supplémentaire.

Pour offrir la propriété de *convergence forte*, la spécification des CRDTs reposent sur la théorie des treillis [22] :

Définition 7 (Spécification des CRDTs)**.** Les CRDTs sont spécifiés de la manière suivante :

(i) Les différents états possibles d'un CRDT forment un sup-demi-treillis, possédant une relation d'ordre partiel $\leq$.

(ii) Les modifications génèrent par inflation un nouvel état supérieur ou égal à l'état original d'après $\leq$.

(iii) Il existe une fonction de fusion qui, pour toute paire d'états, génère l'état minimal supérieur d'après $\leq$ aux deux états fusionnés. Nous parlons alors de borne supérieure ou de Least Upper Bound (LUB) pour catégoriser l'état résultant de cette fusion.

Malgré leur spécification différente, les CRDTs partagent la même sémantique, c.-à-d. le même comportement, et la même interface que les types séquentiels [10] correspondants du point de vue des utilisateur-rices. Ainsi, les CRDTs partagent le comportement des types séquentiels dans le cadre d'exécutions séquentielles. Cependant, ils définissent aussi une sémantique additionnelle pour chaque type de conflit ne pouvant se produire que dans le cadre d'une exécution distribuée.

Plusieurs sémantiques valides peuvent être proposées pour résoudre un type de conflit. Un CRDT se doit donc de préciser quelle sémantique il choisit.

L'autre aspect définissant un CRDT donné est le modèle qu'il adopte pour propager les modifications. Au fil des années, la littérature a établi et défini plusieurs modèles dit de synchronisation, chacun ayant ses propres besoins et avantages. De fait, plusieurs CRDTs peuvent être proposés pour un même type donné en fonction du modèle de synchronisation choisi.

Ainsi, ce qui définit un CRDT est sa ou ses sémantiques en cas de conflits et son modèle de synchronisation. Dans les prochaines sections, nous présentons les différentes sémantiques possibles pour un type donné, l'Ensemble, en guise d'exemple. Nous présentons ensuite les différents modèles de synchronisation proposés dans la littérature, et détaillons leurs contraintes et impact sur les CRDT les adoptant, toujours en utilisant le même exemple.

Matthieu: TODO : Faire le lien avec les travaux de Burckhardt [23] et les MRDTs [24]

10. Nous dénotons comme *types séquentiels* les spécifications usuelles des types de données supposant une exécution séquentielle de leurs modifications.

Chapitre 2

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.

2.1 Résumés des contributions

2.1.1 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 [1]. 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 pour assurer la convergence même en cas de modifications concurrentes. Pour cela, l'approche des CRDTs [3, 4] 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 [25, 26, 27, 28]. 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 [29, 30], soit d'effectuer une GC du surcoût [26, 5, 6]. 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 [5, 6] : 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 [9].

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. De plus, le passage à une représentation interne minimale de la séquence nous permet d'améliorer le coût des modifications suivantes en termes de calculs.

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. Le surcoût induit par ce mécanisme, 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, 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 2.2.1.

(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 [26, 31, 32]. Nous présentons une piste de travail possible pour pallier ce problème dans la sous-section 2.2.2.

2.1.2 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 [33, 34].

Cependant, les applications collaboratives sont historiquement des applications centralisées, e.g. Google Docs [7]. 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 [35]. 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 [13]. 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 [14].

(ii) Les mécanismes d'anti-entropie [36].

(iii) Les protocoles d'établissement de clés de chiffrement de groupe [17].

(iv) Les protocoles d'établissement de topologies réseaux efficientes [16].

(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 connection 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 [37]. 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) [38], 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.

2.2 Perspectives

2.2.1 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.

2.2.2 Détection et fusion manuelle de versions distantes

À l'issue de cette thèse, nous constatons plusieurs limites des mécanismes de résolution de conflits automatiques dans les systèmes P2P à large échelle sujet au churn. La première d'entre elles est l'utilisation d'un contexte causal. Le contexte causal est utilisé par les mécanismes de résolution de conflits 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.

Le contexte causal est utilisé de manière différente en fonction du mécanisme de résolution de conflit. Dans l'approche Operational Transformation (OT), le contexte causal est utilisé par l'algorithme de contrôle pour déterminer les modifications concurrentes à une modification lors de son intégration, afin de prendre en compte leurs effets. Dans l'approche CRDT, le contexte causal est utilisé par la structure de données répliquée à la génération de la modification pour en faire une modification nativement commutative avec les modifications concurrentes, c.-à-d. pour en faire un élément du sup-demi-treillis représentant la structure de données répliquée.

Le contexte causal peut être représenté de différentes manières. Par exemple, le contexte causal peut prendre la forme d'un vecteur de version [39, 40] ou d'un Directed Acyclic Graph (DAG) des modifications [41]. 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 [42]. 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 initiale 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 [43] 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 [44, 45].

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.

2.2.3 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 [46]. Ce dernier propose une synchronisation des noeuds par le biais de différences d'états.

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 LUB [20].

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 [47], 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.

Dans notre comparaison des modèles de synchronisation (cf. **??**, page **??**), nous avons justifié que les CRDTs synchronisés par différences d'états peuvent être utilisés dans les mêmes contextes que les CRDTs synchronisés par états et que les CRDTs synchronisés par opérations. Cette conclusion nous mène à reconsidérer l'intérêt des autres modèles de synchronisation de nos jours.

Par exemple, 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 [48]. 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. [32] propose une solution adaptée aux systèmes P2P à large échelle. Celle-ci se base notamment sur l'utili-

sation de journaux infalsifiables. *Matthieu: TODO : Ajouter refs* Il convient alors d'étudier si l'utilisation de ces structures 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.

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

Baquero et al. [31] proposent un framework pour concevoir des CRDTs synchronisés par opérations : Pure Operation-Based CRDTs. Ce framework a 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 journal partiellement ordonné des opérations, nommé 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.

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 un système P2P dynamique à large échelle sujet 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 [49, 50] 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 [31], 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 une extension du framework Pure Operation-Based CRDTs. 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 [4].
   (ii) Le modèle de livraison utilisé par le système, qui doit être égal ou plus contraint que modèle de livraison minimal du 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 peut être amené à évoluer en fonction de l'état du système et de ses besoins. Par exemple, un système peut 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, le système peut 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

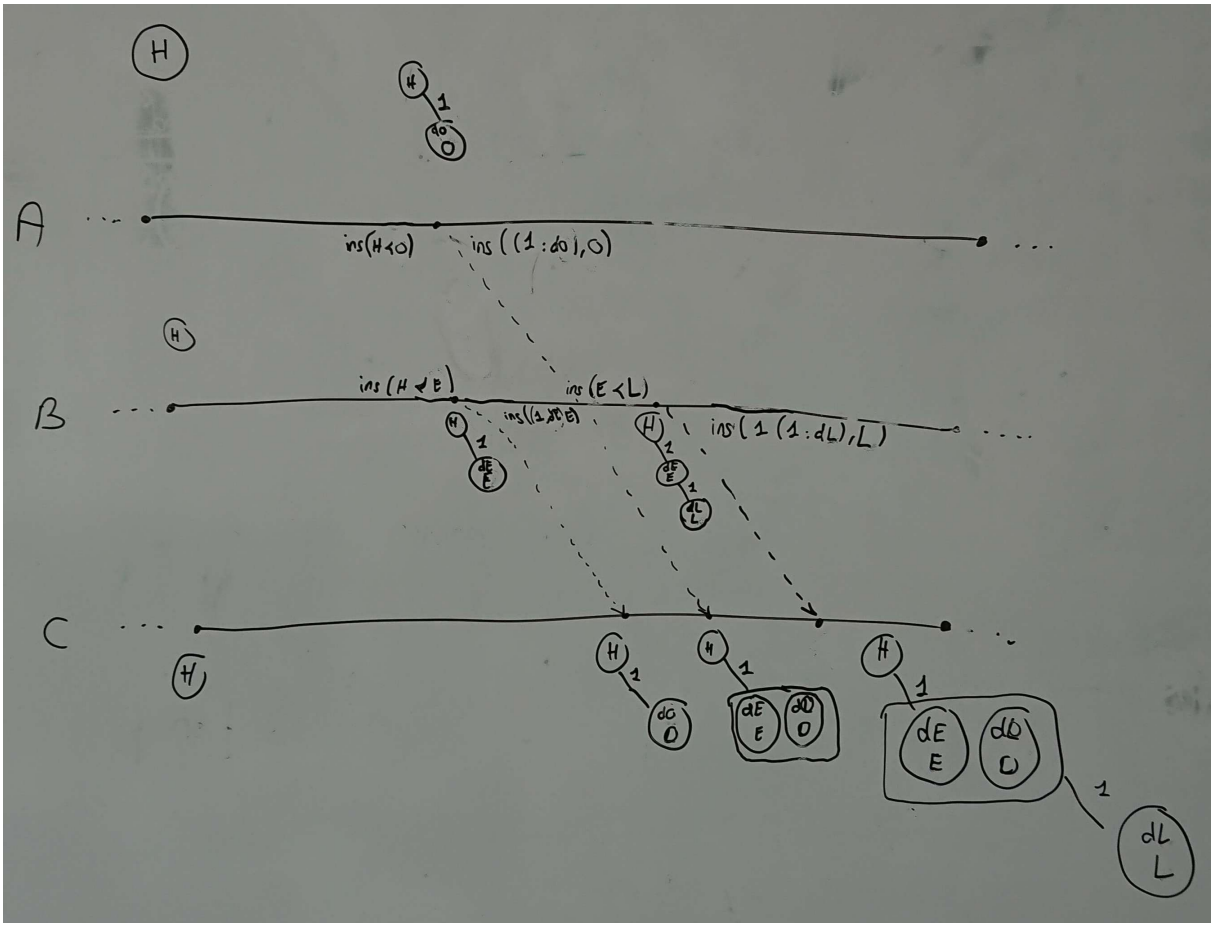

FIGURE A.1 – Modifications concurrentes d'une séquence Treedoc résultant en un entrelacement

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] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] Google. *Google Docs*. URL : https://docs.google.com/.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] Rachid GUERRAOUI, Matej PAVLOVIC et Dragos-Adrian SEREDINSCHI. « Trade-offs in replicated systems ». In : *IEEE Data Engineering Bulletin* 39.ARTICLE (2016), p. 14–26.

[19] Leslie LAMPORT. « Time, Clocks, and the Ordering of Events in a Distributed System ». In : *Commun. ACM* 21.7 (juil. 1978), p. 558–565. ISSN : 0001-0782. DOI : 10.1145/359545.359563. URL : https://doi.org/10.1145/359545.359563.

[20] 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.

[21] 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.

[22] B. A. DAVEY et H. A. PRIESTLEY. *Introduction to Lattices and Order*. 2e éd. Cambridge University Press, 2002. DOI : 10.1017/CBO9780511809088.

[23] Sebastian BURCKHARDT, Alexey GOTSMAN, Hongseok YANG et Marek ZAWIRSKI. « Replicated Data Types : Specification, Verification, Optimality ». In : *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '14. San Diego, California, USA : Association for Computing Machinery, 2014, p. 271–284. ISBN : 9781450325448. DOI : 10.1145/2535838.2535848. URL : https://doi.org/10.1145/2535838.2535848.

[24] Gowtham KAKI, Swarn PRIYA, KC SIVARAMAKRISHNAN et Suresh JAGANNATHAN. « Mergeable Replicated Data Types ». In : *Proc. ACM Program. Lang.* 3.OOPSLA (oct. 2019). DOI : 10.1145/3360580. URL : https://doi.org/10.1145/3360580.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

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

[35] 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.

[36] 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.

[37] OpenRelay. *OpenRelay*. url : https://openrelay.xyz/.

[38] Protocol Labs. *IPFS*. url : https://ipfs.io/.

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

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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