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 16 Décembre 2022

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, LS2N
	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.

Cette évolution a 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. Un cas particulier de ces applications proposent aux utilisateur-rices de travailler ensemble pour la création d'un même contenu, en d'autres termes de collaborer. Nous appelons ces applications des *systèmes collaboratifs* :

Définition 1 (Système collaboratif)**.** Un système collaboratif est un système supportant ses utilisateur-rices dans leurs processus de collaboration pour la réalisation de tâches.

De nos jours, ces systèmes font parties des applications les plus populaires du paysage internet, e.g. la suite logicielle dont fait partie GoogleDocs compte 2 milliards d'utilisateur-rices [1], Wikipedia 788 millions [2], Quora 300 millions [3] ou encore GitHub 60 millions [4]. De leur côté, d'autres plateformes fédèrent leur communautés en organisant ponctuellement des collaborations éphémères impliquant des millions d'utilisateur-rices, e.g. r/Place [5] ou TwitchPlaysPokemon [6].

En raison de leur popularité, les systèmes collaboratifs doivent assurer plusieurs propriétés pour garantir leur bon fonctionnement et qualité de service : une haute disponibilité, tolérance aux pannes et capacité de passage à l'échelle.

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, la plupart de ces systèmes adoptent une architecture décentralisée, que nous illustrons par la Figure 1.1. Dans cette figure, les noeuds aux extrêmités du graphe correspondent à des clients, les noeuds internes à des serveurs et les arêtes du graphe représentent les connexions entre appareils.

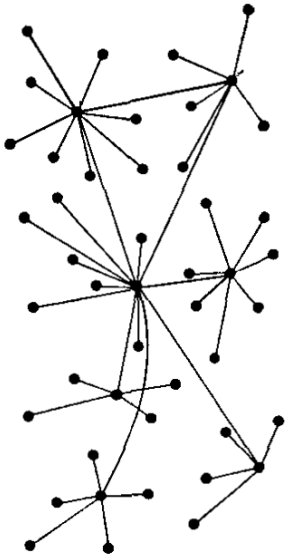


FIGURE 1.1 – Représentation d'une architecture décentralisée [7]

Dans ce type d'architecture, les responsabilités, tâches et la charge travail sont réparties entre un ensemble de serveurs. Malgré ce que le nom de cette architecture peut suggérer, il convient de noter que les serveurs jouent toujours un rôle central dans ces systèmes,. En effet, ces systèmes reposent toujours sur leurs serveurs pour authentifier les utilisateur-rices, stocker leurs données ou encore fusionner les modifications effectuées.

Bien que cette architecture système permette de répondre aux problèmes d'ordre technique que nous présentons précédemment, elle souffre néanmoins de limites. Notamment, de part le rôle prédominant 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.

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.

De plus, les serveurs ne sont pas une ressource libre. En effet, ils sont déployés et maintenus par la ou les organisations qui proposent le système collaboratif. Ces organisations font alors office d'*autorités centrales* du système, e.g. en se portant garantes de l'identité des utilisateur-rices, de l'authenticité d'un contenu ou encore de la disponibilité dudit contenu.

De part le fait que les autorités centrales possèdent les serveurs hébergeant le système, elles ont tout pouvoir sur ces derniers. Ainsi, les utilisateur-rices de systèmes collaboratifs prennent, de manière consciente ou non, le risque que les propriétés présentées précédemment soient transgressées par les autorités auxquelles appartiennent ces applications ou par des tiers avec lesquelles ces autorités interagissent, e.g. des gouvernements. Plusieurs faits d'actualités nous ont malheureusement montré de tels faits, e.g. la censure de Wikipedia par des gouvernements [8], la fermeture de services par les entreprises les proposant [9] ou encore la mise à disposition des données hébergées par des applications aux services de renseignement de différentes nations [10, 11]. Cependant, le coût conséquent de l'infrastructure nécessaire pour déployer des systèmes à large échelle équivalents entrave la mise en place d'alternatives, plus respectueuses de leurs utilisateur-rices.

Ainsi, il nous paraît fondamental de proposer des moyens technologiques rendant accessible la conception et le déploiement des systèmes collaboratifs alternatifs. Ces derniers devraient minimiser le rôle des autorités centrales, voire l'éliminer, de façon à protéger et privilégier les intérêts de leurs utilisateur-rices.

Dans cette optique, une piste de recherche que nous jugeons intéressante est celle des systèmes collaboratifs pair-à-pair (P2P). Cette architecture système, que nous illustrons par la Figure 1.2, place les utilisateur-rices au centre du système et relègue les éventuels serveurs à un simple rôle de support de la collaboration, e.g. la mise en relation des pairs.

Récemment, la conception de systèmes collaboratifs P2P a gagné en traction suite à [12]. Dans cet article, les auteurs définissent un ensemble de propriétés qui correspondent à celles que nous avons établies précédemment, de la Définition 1 à la Définition 8. En

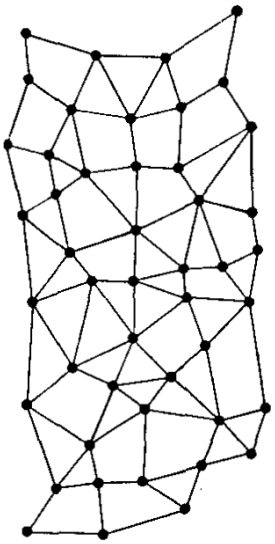


FIGURE 1.2 – Représentation d'une architecture distribuée [7]

utilisant ces propriétés comme critères, les auteurs comparent les fonctionnalités et garanties offertes par les différents types d'applications, notamment les applications lourdes et les applications basées sur le cloud.

La Figure 1.3 détaille le résultat de cette comparaison : alors que les applications basées sur le cloud permettent de nouveaux usages, notamment la collaboration entre utilisateur-rices ou la synchronisation automatique entre appareils, elles retirent à leurs utilisateur-rices toute garantie de pérennité, confidentialité des données et souveraineté des données. Ces dernières propriétés sont pourtant communément offertes par les applications lourdes.

Malgré ce que ce résultat pourrait suggérer, les auteurs affirment que les nouveaux usages offerts par les applications basées sur le cloud ne sont pas antinomiques avec les propriétés de confidentialité, souveraineté, pérennité.

Ainsi, ils proposent un nouveau paradigme de conception d'applications collaboratives P2P, nommées Local-First Software (LFS). Ce paradigme vise à la conception d'applications offrant le meilleur des approches existantes, c.-à-d. des applications cochant l'intégralité des critères de la Figure 1.3. Nous partageons cette vision.

Cependant, de nombreuses problématiques de recherche identifiées dans [12] sont encore non résolues et entravent la démocratisation des applications LFS, notamment celles à large échelle. Spécifiquement, les applications LFS se doivent de répliquer les données entre les 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.

Toutefois, compte tenu des propriétés visées par les applications LFS, plusieurs contraintes restreignent le choix des méthodes de réplication possibles. Ainsi, pour permettre le fonc-

Technology	Section	1. Fast (§ 2.1)	2. Multi-device (§ 2.2)	3. Offline (§ 2.3)	4. Collaboration (§ 2.4)	5. Longevity (§ 2.5)	6. Privacy (§ 2.6)	7. User control (§ 2.7)
Applications employed by end users								
Files + email attachments	§ 3.1.1	✓	–	✓	●	✓	–	✓
Google Docs	§ 3.1.2	–	✓	–	✓	–	●	–
Trello	§ 3.1.2	–	✓	–	✓	–	●	●
Pinterest	§ 3.1.2	●	✓	●	✓	●	●	●
Dropbox	§ 3.1.3	✓	–	–	●	✓	–	✓
Git + GitHub	§ 3.1.4	✓	–	✓	–	✓	–	✓
Technologies employed by application developers								
Thin client (web apps)	§ 3.2.1	●	✓	●	✓	●	●	●
Thick client (mobile apps)	§ 3.2.2	✓	–	✓	●	–	●	●
Backend-as-a-service	§ 3.2.3	–	✓	✓	–	●	●	●
CouchDB	§ 3.2.4	–	–	✓	●	–	–	–

FIGURE 1.3 – Évaluation d'applications et de technologies vis-à-vis des 7 propriétés visées par les applications Local-First Softwares [12]. ✓, – et ● indiquent respectivement que l'application ou la technologie satisfait pleinement, partiellement ou aucunement le critère évalué.

tionnement en mode hors-ligne de l'application, c.-à-d. la consultation et la modification de contenu, les applications LFS doivent obligatoirement relaxer la propriété de cohérence des données.

Définition 9 (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.

Ainsi, les applications LFS doivent autoriser les noeuds possédant une copie de la donnée à diverger temporairement, c.-à-d. à posséder des copies dans des états différents à un moment donné. Pour permettre cela, les applications LFS doivent adopter des méthodes de réplication dites optimistes [13], c.-à-d. qui permettent la consultation et la modification de la donnée sans coordination au préalable avec les autres noeuds [1]. Un mécanisme de synchronisation permet ensuite aux noeuds de partager les modifications effectuées et de les intégrer de façon à converger à terme [14], c.-à-d. obtenir de nouveau des états équivalents.

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

1. Les méthodes de réplication optimistes s'opposent aux méthodes de réplication dites pessimistes qui nécessitent une coordination préalable entre les noeuds avant toute modification de la donnée et interdisent ainsi toute divergence

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 que nous visons, c.-à-d. des applications LFS à large échelle, limitent les choix possibles concernant les mécanismes de résolution de conflits. Notamment, ces applications 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. Ce nombre de noeuds peut croître de manière non-bornée. Les mécanismes de résolution de conflits choisis devraient donc rester efficaces, indépendamment de l'évolution de ce paramètre.

De plus, les noeuds composant le système 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* [15]. Ainsi, de part l'absence de garantie sur le nombre de noeuds connectés de manière stable, les applications LFS à large échelle 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. des mécanismes nécessitant une communication ininterrompue entre un ensemble de noeuds du système pour prendre une décision, e.g. des algorithmes de consensus [16, 17].

Ainsi, pour permettre la conception d'applications LFS à large échelle, il convient de disposer de mécanismes de résolution de conflits pour l'ensemble des types de données avec une complexité algorithmique efficace peu importe le 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) [2] [18, 19] sont des nouvelles spécifications des types de données usuels, e.g. l'Ensemble ou la Séquence. Ils sont conçus pour permettre à un ensemble de noeuds d'un système de répliquer une donnée et pour leur permettre de la consulter, de la modifier sans aucune coordination préalable et d'assurer à terme la convergence des copies. 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.

2. Conflict-free Replicated Data Type (CRDT) : Type de données répliquées sans conflits

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 d'intégrer les modifications, 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, [20, 21] 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 système.

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 problématique, nous proposons dans cette thèse 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. Finalement, nous définissons un mécanisme de Garbage Collection (GC) [3] des métadonnées du mécanisme de renommage pour supprimer à terme son propre surcoût. De cette manière, nous proposons un CRDT pour le type Séquence dont le surcoût est périodiquement réduit, tout en 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

Comme évoqué précédemment, la conception d'applications LFS à large échelle présente un ensemble de problématiques issues 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 ?

3. Garbage Collection (GC) : Récupération de la mémoire

(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 LFS à large échelle, sûre et sans autorités centrales ?*

Pour étudier cette problématique, l'équipe Coast développe l'application Multi User Text Editor (MUTE) [4] [22]. Il s'agit d'un Proof of Concept (PoC) [5] d'éditeur de texte web collaboratif P2P temps réel chiffré de bout en bout. Ce projet permet à l'équipe de présenter ses travaux de recherche portant sur les mécanismes de résolutions de conflits automatiques pour le type Séquence [23, 24, 25] et les mécanismes d'authentification des pairs dans les systèmes sans autorités centrales [26, 27].

De plus, en inscrivant ses travaux dans le cadre d'un système complet, ce projet permet à l'équipe d'identifier de nouvelles problématiques en relation avec les nombreux domaines de recherche nécessaires à la conception d'un tel système, e.g. le domaine des protocoles d'appartenance aux groupes [28, 29], des topologies réseaux P2P [30] ou encore des protocoles d'établissement de clés de chiffrement de groupe [31].

Dans le cadre de cette thèse, nous avons contribué au développement de ce projet. Nous avons notamment implémenté plusieurs CRDTs pour le type Séquence [23, 25] et le protocole d'appartenance au réseau SWIM [28].

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 CRDTs et plus particulièrement celui des CRDTs pour le type Séquence. 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 réaliser 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

4. Disponible à l'adresse : `https://mutehost.loria.fr`
5. Proof of Concept (PoC) : Preuve de concept

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.

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 [32, 33].

Plusieurs outils d'édition collaborative centralisés basés sur l'approche Operational Transformation (OT) [34] ont permis de populariser l'édition collaborative temps réel de texte [35, 36]. Ces approches souffrent néanmoins de leur architecture centralisée. Notamment, ces solutions rencontrent des difficultés à passer à l'échelle [37, 38] et posent des problèmes de confidentialité [11, 10].

L'approche CRDT offre une meilleure capacité de passage à l'échelle et est compatible avec une architecture P2P [39]. Ainsi, de nombreux travaux [40, 41, 42] 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 (LFSs) [12, 43]. Ce paradigme vise le développement d'applications collaboratives, P2P, pérennes et rendant la souveraineté de leurs données aux utilisateurs.

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) [22] [6] [7], 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 [23] et RenamableLogootSplit [25] 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 [28].

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 sections 2.3, et 2.4.

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 :

6. Disponible à l'adresse : `https://mutehost.loria.fr`
7. 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

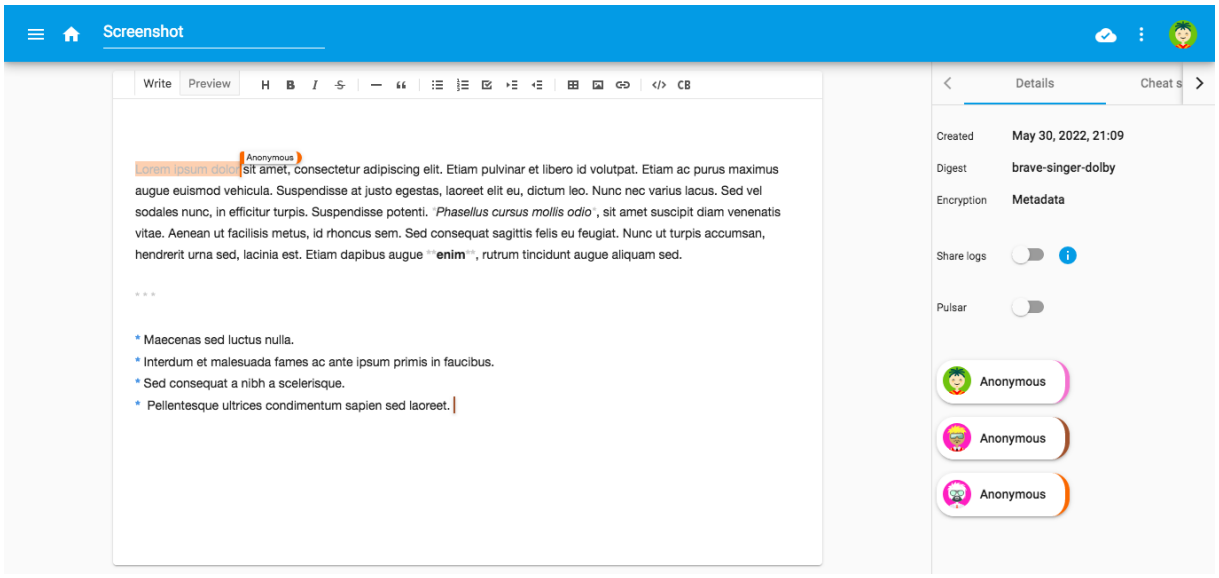


FIGURE 2.1 – Capture d'écran d'une session d'édition collaborative avec MUTE

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 10 (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 11 (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.

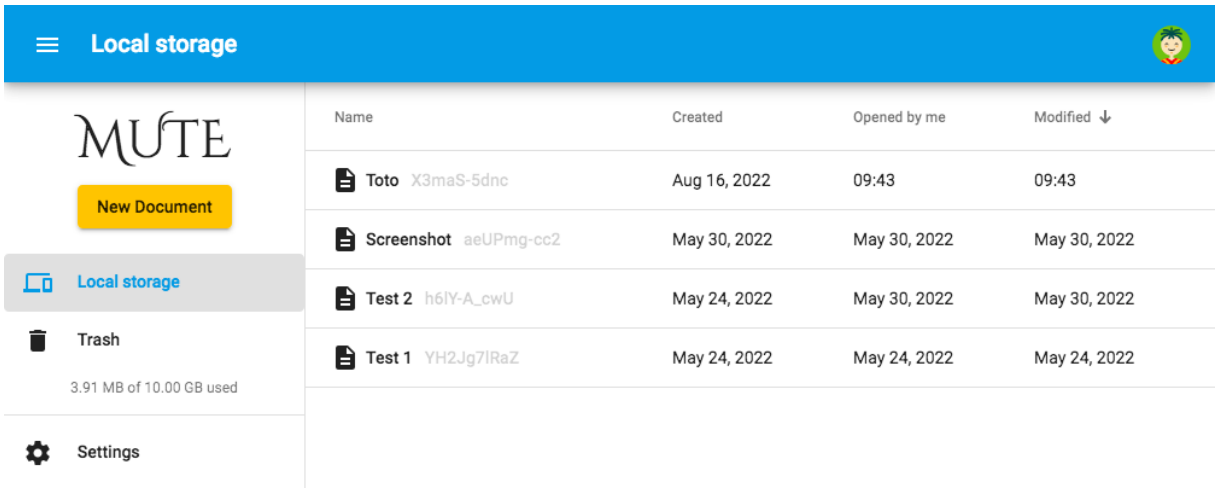


FIGURE 2.2 – Capture d'écran de la liste des documents.

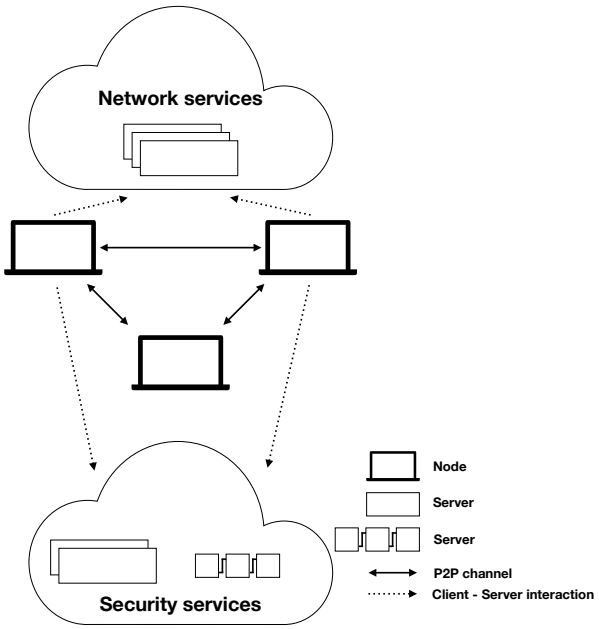


FIGURE 2.3 – Architecture système de l'application MUTE

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 2.5.

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 2.6 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.

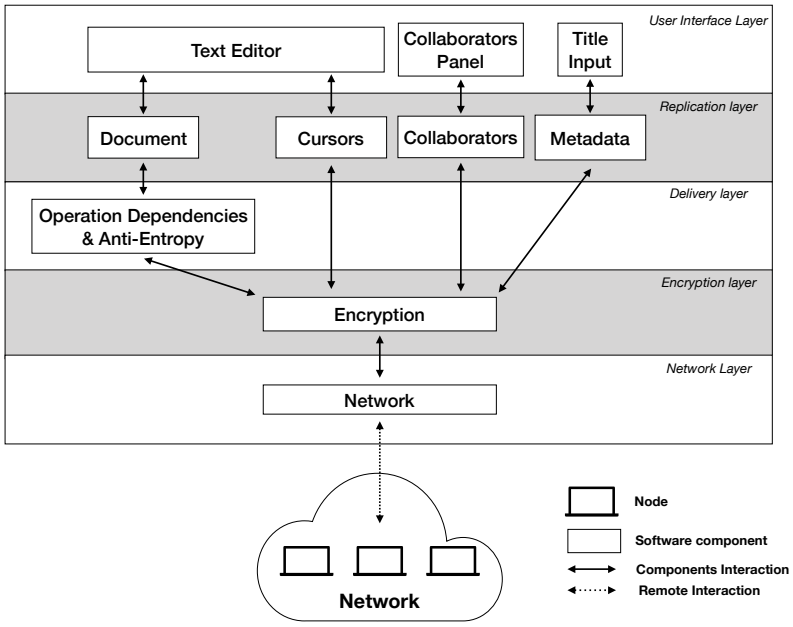


FIGURE 2.4 – Architecture logicielle de l'application MUTE

(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 principalement d'un éditeur de texte. Ce dernier supporte le langage de balisage Markdown [44]. Ainsi, l'éditeur permet d'inclure plusieurs éléments légers de style. Les balises du langage Markdown étant du texte, elles sont répliquées nativement par le CRDT utilisé en interne par MUTE pour représenter la séquence.

L'interface de la page de l'éditeur de document est agrémentée de plusieurs mécanismes permettant d'établir une conscience de groupe entre les collaborateur-rices. L'indicateur en haut à droite de la page représente le statut de connexion de l'utilisateur-rice. Celui-ci permet d'indiquer à l'utilisateur-rice s'iel est actuellement connecté-e au réseau P2P, en cours de connexion, ou si un incident réseau a lieu.

De plus, MUTE affiche sur la droite de l'éditeur la liste des collaborateur-rices actuellement connecté-es. Un curseur ou une sélection distante est associée pour chaque membre de la liste. Ces informations permettent d'indiquer à l'utilisateur-rice dans quelles sections du document ses collaborateur-rices sont en train de travailler. Ainsi, iels peuvent se répartir la rédaction du document de manière implicite ou suivre facilement les modifications d'un-e collaborateur-rice.

Bien que fonctionnelle, cette interface souffre néanmoins de plusieurs limites. Notamment, nous n'avons pas encore pu étudier la littérature concernant les mécanismes de conscience pour supporter la collaboration, au-delà du système de curseurs distants.

Nous identifions ainsi plusieurs axes de travail pour ces mécanismes. Tout d'abord, l'axe des *mécanismes de conscience des changements*. Le but serait de proposer des mécanismes pour :

(i) Mettre en lumière de manière intelligible les modifications effectuées par les collabateur-rices dans le cadre de collaborations temps réel à large échelle. Un tel mécanisme représente un défi de part le débit important de changements, potentiellement à plusieurs endroits du document de manière quasi-simultanée, à présenter à l'utilisateur-rice.

(ii) Mettre en lumière de manière intelligible les modifications effectuées par les collaborateur-rices dans le cadre de collaborations asynchrones. De nouveau, ce mécanisme représente un défi de part la quantité massive de changements, une fois encore potentiellement à plusieurs endroits du document, à présenter à l'utilisateur-rice.

Une piste de travail potentiellement liée serait l'ajout d'une fonctionnalité d'historique du document, permettant aux utilisateur-rices de parcourir ses différentes versions obtenues au fur et à mesure des modifications. L'intégration d'une telle fonctionnalité dans un éditeur P2P pose cependant plusieurs questions : quel historique présenter aux utilisateur-rices, sachant que chacun-e a potentiellement observé un ordre différent des modifications ? Doit-on convenir d'une seule version de l'historique ? Dans ce cas, comment choisir et construire cet historique ?

Le second axe de travail sur les mécanismes de conscience concerne les *mécanismes*

de conscience de groupe. Actuellement, nous affichons l'ensemble des collaborateur-rices actuellement connecté-es. Cette approche s'avère lourde voire entravante dans le cadre de collaborations à large échelle où le nombre de collaborateur-rices dépasse plusieurs centaines. Il convient donc de déterminer quelles informations présenter à l'utilisateur-rice dans cette situation, e.g. une liste compacte de pairs et leur curseur respectif, ainsi que le nombre de pairs total.

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 [8] (cf. **??**, page **??**), soit de RenamableLogootSplit[8] (cf. **??**, page **??**) ou soit de Dotted LogootSplit [9] [24]. 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 texte, c.-à-d. de messages au format $\langle ins, index, elts \rangle$ ou $\langle rmv, index, length \rangle$. Lorsque l'utilisateur effectue des modifications locales, celles-ci sont détectées par l'éditeur et mises sous la forme d'opérations texte. 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 texte 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 donnée additionnelle, nous associons un Last-Writer-Wins Register CRDT synchronisé par états [19] au document. De façon similaire, nous utilisons un First-Writer-Wins Register CRDT synchronisé par états pour représenter la date de création du document.

L'utilisation de ces structures de données nous permet donc de représenter le document texte ainsi que ses métadonnées. Nous identifions cependant plusieurs axes d'évolution pour cette couche. La première d'entre elles concerne l'ajout de styles au document. Comme indiqué dans la section 2.2, nous utilisons le langage de balisage Markdown pour inclure plusieurs éléments de style. Cette solution a pour principal intérêt de reposer sur du texte qui s'intègre directement dans le contenu du document. Ainsi, les balises de style sont répliquées nativement avec le contenu du document par le CRDT représentant

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

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

FIGURE 2.5 – Entrelacement de balises Markdown produisant une anomalie de style

ce dernier. Cette solution montre cependant ses limites lorsque plusieurs éléments styles sont ajoutés sur des zones de texte se superposant, notamment en concurrence. Les balises Markdown produites sont alors entrelacées. La figure Figure 2.5 illustre un tel exemple.

Dans cet exemple, le texte "diam venenatis" a été mis en gras tandis que le texte "venenatis vitae" a été mis en italique. Le résultat attendu est donc "**diam *venenatis* *vitae***". Il ne s'agit toutefois pas du résultat affiché par notre éditeur de texte, la syntaxe du langage Markdown n'étant pas respectée. Les utilisateur-rices doivent donc manuellement corriger l'anomalie de style engendrée.

Afin de prévenir ce type d'anomalie, il conviendrait d'intégrer un CRDT pour représenter le style du document, tel que Peritext [45]. Un type de donnée dédié nous permettrait de plus d'inclure des effets de style non-disponibles dans le langage Markdown, e.g. la mise en page ou encore l'utilisation de couleurs.

Une second piste d'évolution consiste à rendre possible l'intégration et l'édition collaborative d'autres types de contenu que du texte au sein d'un document MUTE, e.g. des listes de tâches, des feuilles de calcul ou encore des schémas. L'évolution d'éditeur collaboratif de documents texte à éditeur collaboratif de documents multimédia élargirait ainsi les contextes d'utilisation de MUTE.

Cette piste de recherche nécessite de concevoir et d'intégrer des CRDTs pour chaque type de document supplémentaire. Elle nécessite aussi la conception d'un mécanisme assurant la composition de ces CRDTs, c.-à-d. la conception d'un méta-CRDT. Ce méta-CRDT devrait assurer plusieurs responsabilités. Tout d'abord, il devrait permettre l'ajout et la suppression de CRDTs au sein du document multimédia, ainsi que leur ré-ordonnement. Ensuite, le méta-CRDT devrait définir les sémantiques de résolution de conflits employées en cas de modifications concurrentes sur le document, e.g. la modification du contenu d'un des CRDTs du document en concurrence de la suppression dudit CRDT. Finalement, le méta-CRDT devrait proposer différents modèles de cohérence à l'application. Il devrait par exemple proposer de garantir le modèle de cohérence causal, c.-à-d. d'ordonner les modifications sur le méta-CRDT et les CRDTs qui composent le document en utilisant la relation *happens-before* (cf. **??**, page **??**).

2.3.2 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.

MUTE présente 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 [28]. Ce protocole d'appartenance au réseau offre les propriétés intéressantes suivantes. Tout d'abord, le nombre de messages diffusés sur le réseau est proportionnel linéairement 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 protocole. 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 2.6.

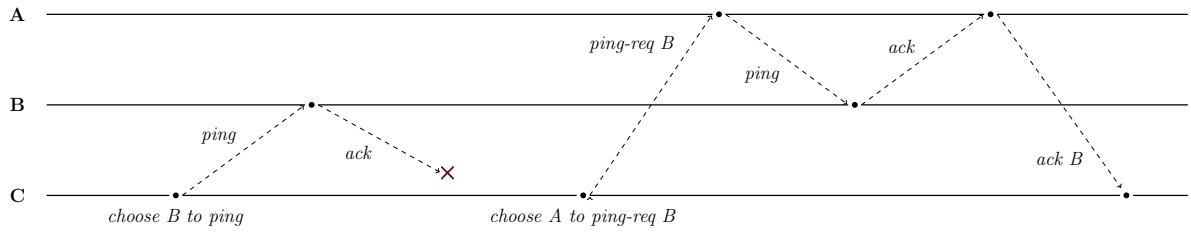


FIGURE 2.6 – Exécution du mécanisme de détection des défaillances par le noeud C pour tester le noeud B

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 devrait prendre 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 de 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.

Afin de représenter la liste des collaborateur-rices, le protocole SWIM utilise la structure de données présentée par la Définition 12 :

Définition 12 (Liste des collaborateur-rices)**.** La *liste des collaborateur-rices* est un ensemble de triplets $\langle nodeId, nodeStatus, nodeIncarn \rangle$ avec :

(i) *nodeId*, l'identifiant du noeud correspondant à ce tuple.

(ii) *nodeStatus*, le 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.

(iii) *nodeIncarn*, le 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 [10] 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.

10. [28] montre que choisir une valeur constante faible comme λ suffit néanmoins à garantir la dissémination des mises à jour à l'ensemble du réseau.

Chaque mise à jour du groupe est stockée dans une liste et se voit attribuer un compteur entier, initialisé avec $\lambda \log n$ où n est 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 2.3.2. 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 [28]. Notre première modification porte sur l'ordre de priorité entre les états d'un pair.

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 d'anti-entropie : 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 les défaillances éventuelles du réseau. Ainsi, dans les faits, nous avons mis en place un CRDT synchronisé par différences d'états (cf. **??**, page **??**) 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 [28]. 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 ont été proposées dans [29]. 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.

2.3.3 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 livraison

Comme indiqué précédemment, la couche livraison est formée d'un unique composant, que nous nommons module de livraison. Ce module est associé aux CRDTs synchronisés par opérations représentant le document texte, c.-à-d. LogootSplit ou RenamableLogootSplit.

Le rôle de ce module est de garantir que le modèle de livraison des opérations requis par le CRDT pour assurer la convergence à terme (cf. **??**, page **??**) soit satisfait, c.-à-d. que l'ensemble des opérations soient livrées dans un ordre correct à l'ensemble des noeuds.

Pour cela, le module de livraison doit implémenter les contraintes imposées par ces CRDTs sur l'ordre de livraison des opérations (cf. **??**, page **??** et **??**, page **??**). Pour rappel, le modèle de livraison de RenamableLogootSplit est le suivant :

(i) Une opération doit être livrée à l'ensemble des noeuds à terme.

(ii) Une opération doit être livrée qu'une seule et unique fois aux noeuds.

(iii) Une opération *remove* doit être livrée à un noeud une fois que les opérations *insert* des éléments concernés par la suppression ont été livrées à ce dernier.

(iv) Une opération peut être délivrée à un noeud qu'à partir du moment où l'opération *rename* qui a introduit son époque de génération a été délivrée à ce même noeud.

Nous décrivons ci-dessous comment nous assurons chacune de ces contraintes.

2.4.1 Livraison des opérations en exactement un exemplaire

Afin de respecter la contrainte de livraison en exactement un exemplaire, il est nécessaire d'identifier de manière unique chaque opération. Pour cela, le module de livraison ajoute un *Dot* [46] à chaque opération :

Définition 13 (Dot)**.** Un *Dot* est une paire $\langle nodeId, nodeSyncSeq \rangle$ où

(i) *nodeId*, l'identifiant unique du noeud qui a généré l'opération.

(ii) *nodeSyncSeq*, 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 de livraison 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 version. Cette structure nous permet de réduire à un dot par noeud le surcoût en métadonnées du module de livraison, 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 2.7 illustre son fonctionnement.

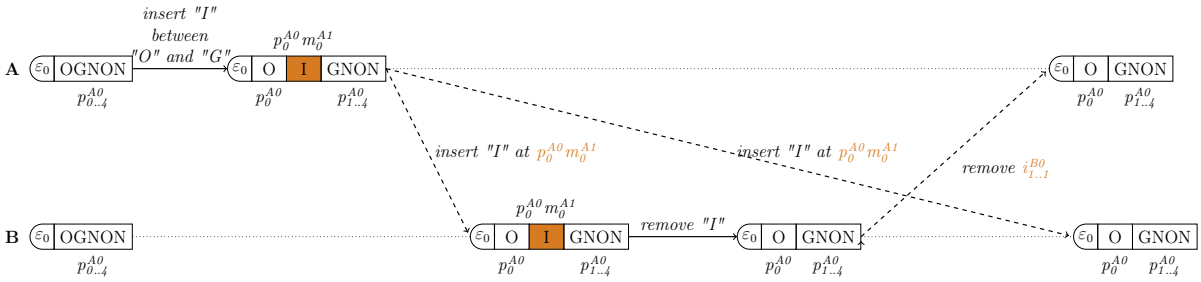
Dans cet exemple, qui reprend celui de la **??**, deux noeuds A et B répliquent une séquence. Initialement, celle-ci contient les éléments "OGNON". Ces éléments ont été insérés un par un par le noeud A, donc par le biais des opérations *a1* à *a5*. Le module de livraison de chaque noeud maintient donc 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 de livraison et est envoyée au noeud B. À la réception de cette opération, le module de B compare son dot avec son vecteur de version local. 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.

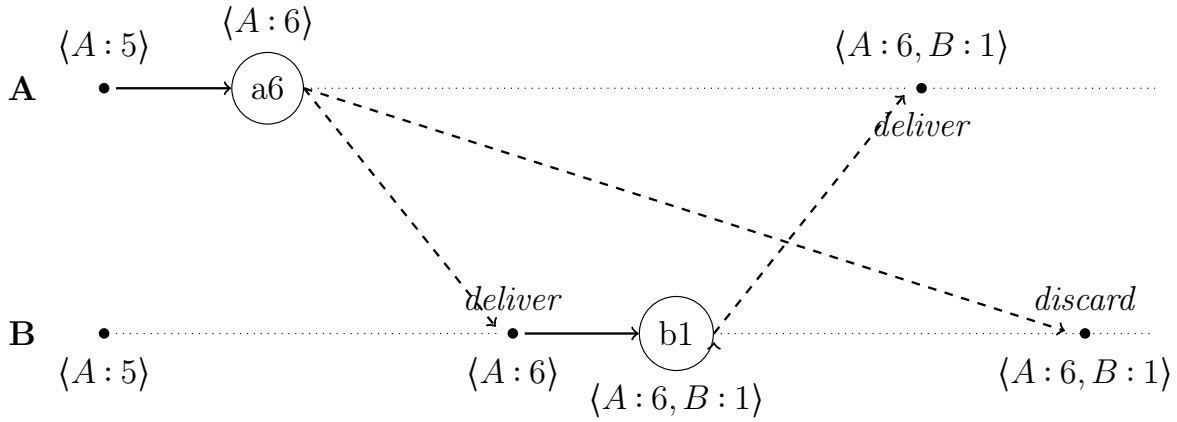
Le noeud B supprime ensuite l'élément nouvellement inséré. S'agissant de la première modification de B, cette modification *b1* ajoute l'entrée correspondante dans le vecteur de version $\langle A : 6, B : 1 \rangle$. L'opération est envoyée au noeud A. Cette opération étant la prochaine opération attendue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

Finalement, le noeud B reçoit de nouveau l'opération *a6*. Son module de livraison 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* est donc mise en attente jusqu'à la livraison de l'opération *b2*.



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



(b) État et comportement du module de livraison au cours de l'exécution décrite en Figure 2.7a

FIGURE 2.7 – Gestion de la livraison en exactement un exemplaire des opérations

Ainsi, l'implémentation de livraison en exactement un exemplaire d'une opération 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 en exactement un exemplaire 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* [47]. 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 *dans le désordre* des opérations, c.-à-d. une livraison des opérations dans un ordre différent de leur ordre d'émission, tout en garantissant une livraison en exactement un exemplaire et en compactant efficacement les données stockées par le module de livraison à terme.

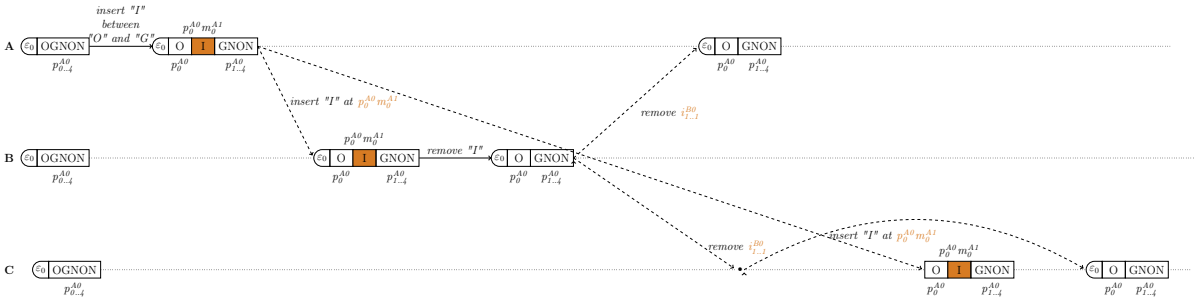
2.4.2 Livraison de l'opération *remove* après l'opération *insert*

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.

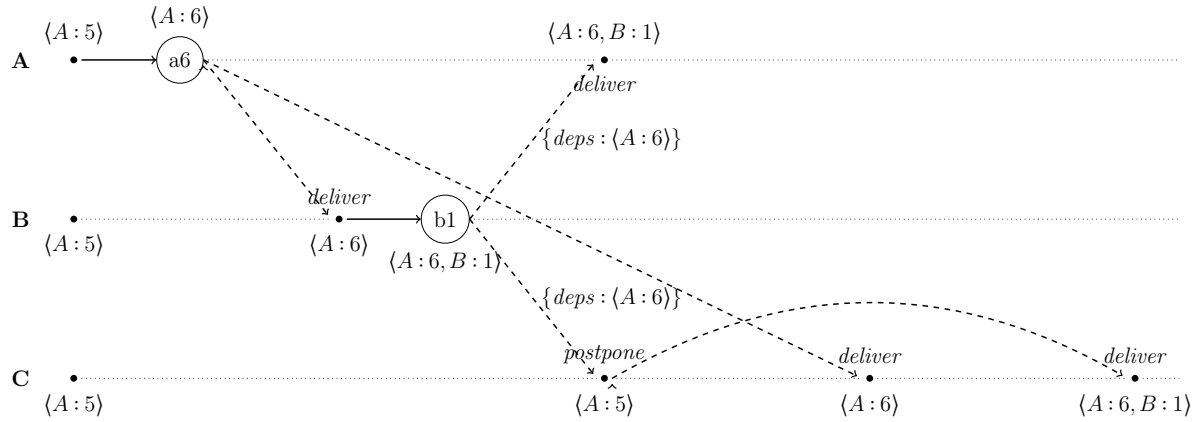
Pour cela, le module de livraison ajoute un ensemble *Deps* à chaque opération *remove* avant de la diffuser :

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 2.8.



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



(b) État et comportement du module de livraison au cours de l'exécution décrite en Figure 2.8a

FIGURE 2.8 – Gestion de la livraison des opérations *remove* après les opérations *insert* correspondantes

Cet exemple reprend et complète celui de la **??**. 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 de livraison calcule ses dépendances avant de la diffuser. Pour chaque élément supprimé ("I"), le module de livraison récupère l'identifiant de l'élément ($p_0^{A0} m_0^{A1}$) et en extrait l'identifiant du noeud qui l'a inséré (A). Le module 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 de livraison 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 livrée. Le module de livraison 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 inséré les éléments supprimés. Pour cela, il serait nécessaire de parcourir le journal des opérations à la recherche des opérations *insert* correspondantes. Cette méthode serait plus coûteuse, sa complexité dépendant du nombre d'opérations dans le journal des opérations, et incompatible avec un mécanisme tronquant le journal 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.

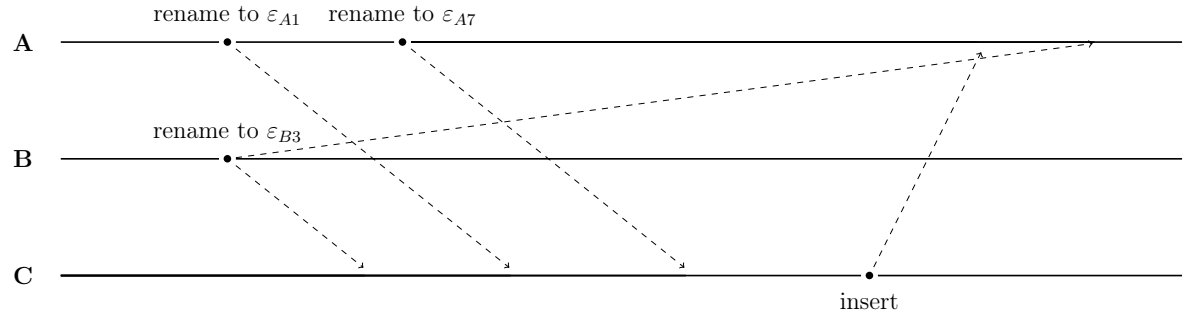
2.4.3 Livraison des opérations après l'opération *rename* introduisant leur époque

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

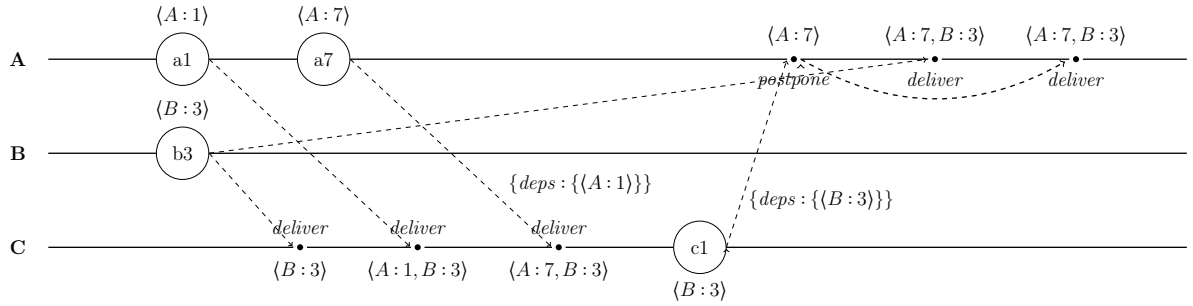
Pour cela, le module de livraison doit donc récupérer l'époque courante de la séquence répliquée, récupérer le dot de l'opération *rename* l'ayant introduite et l'ajouter en tant que dépendance de chaque opération. Cependant, dans notre implémentation, le module de livraison et le module représentant la séquence répliquée sont découplés et ne peuvent interagir directement l'un avec l'autre.

Pour remédier à ce problème, le module de livraison 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. Le module conserve alors seule-

ment 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 2.9 illustre ce fonctionnement.



(a) Exécution



(b) État et comportement du module de livraison au cours de l'exécution décrite en Figure 2.9a

FIGURE 2.9 – Gestion de la livraison des opérations après l'opération *rename* qui introduit leur époque

Dans la Figure 2.9a, 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 2.9b, nous faisons apparaître l'état du module de livraison 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 de livraison 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 de livraison 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 de livraison 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 le module de livraison, 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.

2.4.4 Livraison des opérations à terme

La contrainte restante du modèle de livraison précise que toutes les opérations doivent être livrées à l'ensemble des 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 [48]. 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 journal 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 2.10 illustre ce mécanisme.

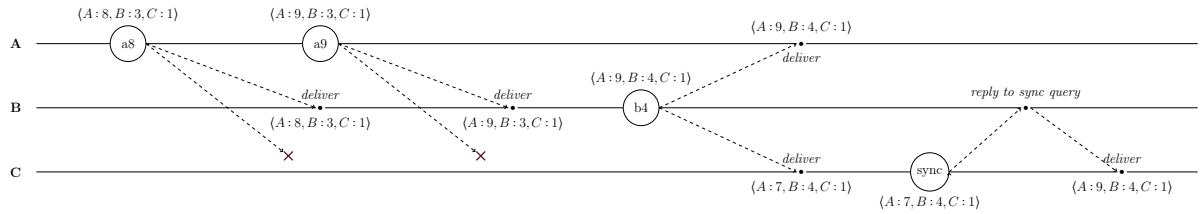


FIGURE 2.10 – Utilisation du mécanisme d'anti-entropie par le noeud C pour se synchroniser avec le noeud B

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 journal 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 en exactement un exemplaire, comme présenté en sous-section 2.4.1. 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 journal 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 journal 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 journal comme évoqué en **??**, ce journal 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.

2.5 Couche réseau

Pour permettre aux différents noeuds de communiquer, MUTE repose sur la librairie Netflux [11]. Développée au sein de l'équipe Coast, cette librairie permet de construire un réseau P2P entre des navigateurs, mais aussi des agents logiciels.

2.5.1 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 [12] 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 2.11, nous représentons une collaboration réalisée avec MUTE, composé de noeuds formant un réseau P2P, de différents serveurs nécessaires à la mise en place du réseau P2P. Finalement, nous représentons les interactions entre les noeuds et ces serveurs.

Nous décrivons ci-dessous le rôle respectif de chaque type de serveur 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

11. `https://github.com/coast-team/netflux`
12. Application Programming Interface (API) : Interface de Programmation

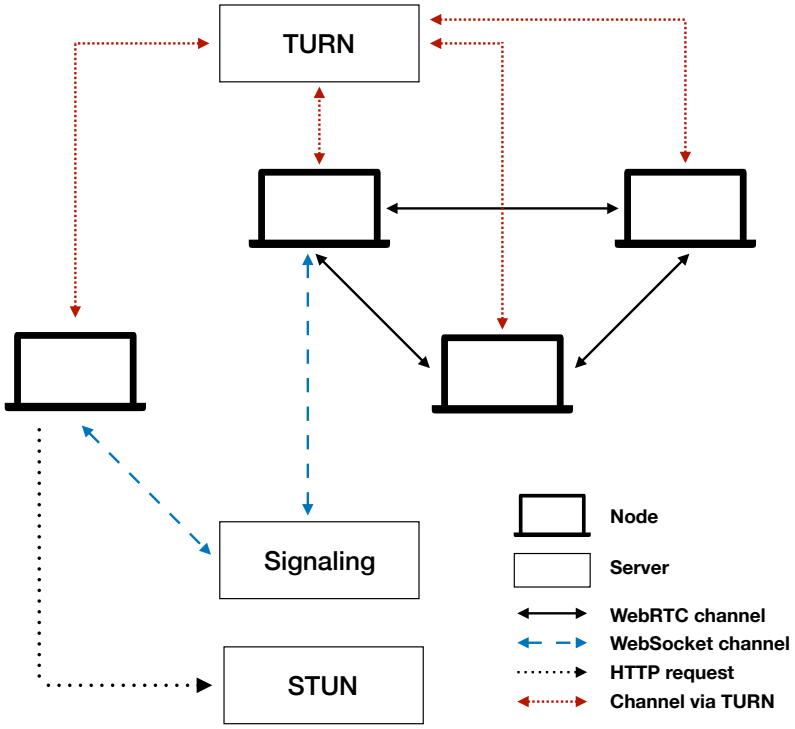


FIGURE 2.11 – Architecture système pour la couche réseau de MUTE

est mis en relation avec un noeud du réseau P2P par son intermédiaire et échange les 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.

Rôles 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.

2.5.2 Topologie réseau et protocole de diffusion des messages

Netflux établit un réseau P2P par document. Chacun de ces réseaux est un réseau entièrement maillé : chaque noeud se connecte à l'ensemble des autres noeuds. Nous illustrons cette topologie par la Figure 2.12.

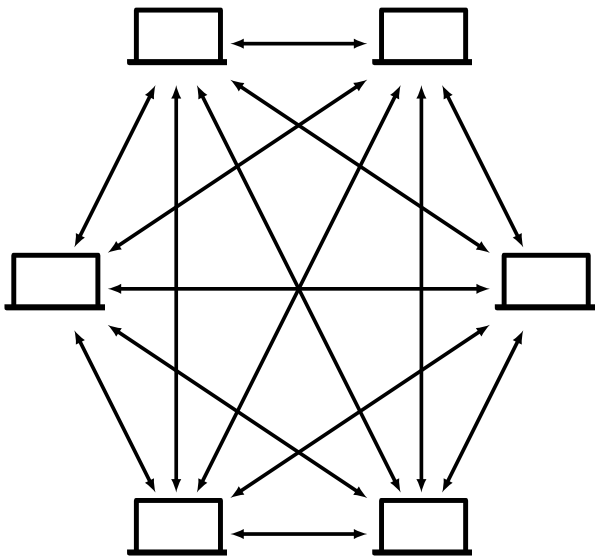


FIGURE 2.12 – Topologie réseau entièrement maillée

Cette topologie présente les avantages suivants :

(i) Sa simplicité.

(ii) Le nombre de sauts entre les noeuds, minimal, qui permet de minimiser le délai de communication.

(iii) La redondance de ses routes, qui permet aux noeuds de continuer à communiquer même en cas de défaillance d'une ou plusieurs routes.

Cependant, cette topologie est limitée par sa capacité de passage à l'échelle. En effet, elle implique un nombre de connexions qui dépend linéairement du nombre de noeuds. Chacune de ces connexions impliquant un coût, cette topologie n'est adaptée qu'à des groupes de taille réduite.

De plus, nous associons à cette topologie réseau un protocole de diffusion des messages tout aussi simple : lorsqu'un noeud effectue une modification, il diffuse le message correspondant à l'ensemble des noeuds, c.-à-d. il envoie une copie du message à chacun des noeuds. La charge de travail pour la diffusion d'un message est donc assumée uniquement par son auteur, ce qui s'avère prohibitif dans le cadre de collaborations à large échelle.

Afin de supporter des collaborations à large échelle, il est donc nécessaire de mettre en place :

(i) Une topologie limitant le nombre de connexions par noeud.

(ii) Un protocole de diffusion des messages répartissant la charge entre les noeuds.

Le protocole d'échantillonnage aléatoire de pairs adaptatif Spray [30] répond à notre première limite. Ce protocole permet d'établir un réseau P2P connecté. Toutefois, la topologie adoptée limite le voisinage de chaque noeud, c.-à-d. le nombre de connexions que chaque noeud possède, à un facteur logarithmique par rapport au nombre total de noeuds. De plus, ce protocole est adapté à un réseau P2P dynamique, c.-à-d. il ajuste le voisinage des noeuds au fur et à mesure des connexions et déconnexions des noeuds.

La topologie résultante est propice à l'emploi d'un protocole de diffusion épidémique des messages, tel que celui utilisé par SWIM (cf. section 2.3.2, page 22). Pour rappel, ce type de protocole consiste à ce qu'un noeud ne diffuse un message qu'à un sous-ensemble de ses voisins. À la réception de son message, ses voisins diffusent à leur tour le message à une partie de leur voisinage, et ainsi de suite. L'emploi de ce type de protocole permet ainsi de répartir entre les noeuds la charge de travail nécessaire à la diffusion du message à l'ensemble des noeuds.

Ces modifications rendraient donc viables les collaborations à large échelle sur MUTE. En contrepartie, le délai nécessaire pour la diffusion d'une modification augmenterait, c.-à-d. le temps nécessaire pour qu'une modification effectuée par un noeud soit intégrée par les autres noeuds. Il s'agit toutefois d'un compromis que nous jugeons nécessaire.

2.6 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 [31]. Il nécessite que chaque noeud possède une paire de clés de chiffrement et enregistre sa clé publique auprès

d'un PKI [13].

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 [26, 27]. 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 2.13.

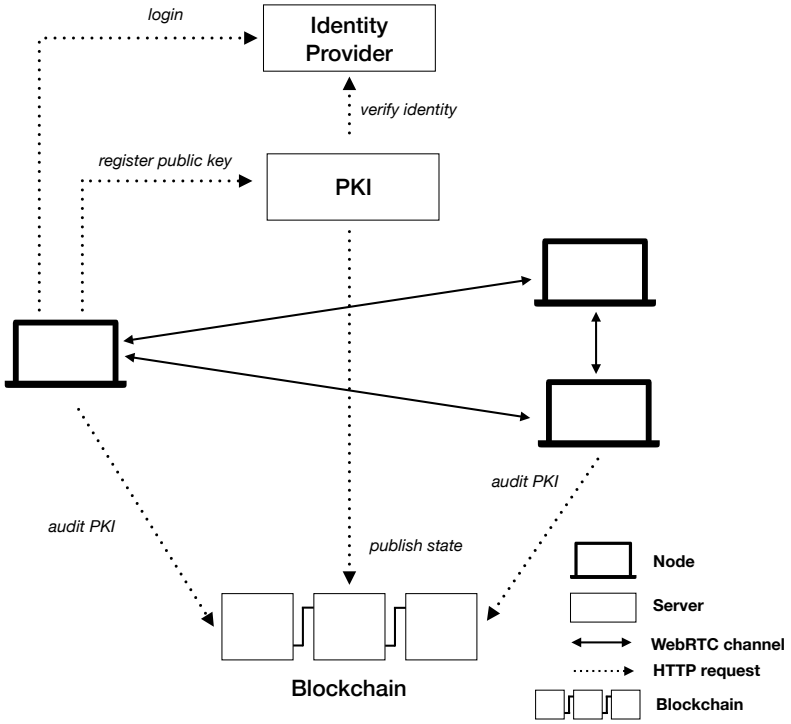


FIGURE 2.13 – Architecture système pour la couche sécurité de MUTE

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 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 à chaque évolution des noeuds connectés : à chaque connexion ou déconnexion d'un noeud, une nouvelle clé est recalculée avec les collaborateur-rices présents. Ce facteur de changement de la clé de chiffrement, nécessaire pour garantir la *backward secrecy* et *forward secrecy* (cf. Définition 10, page 14 et Définition 11, page 14), induit plusieurs problèmes.

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

Tout d'abord, ce facteur nous empêche de réutiliser cette même clé de chiffrement pour mettre en place un mécanisme de stockage des opérations chiffrées chez un ou des tiers, e.g. sur des noeuds du réseau P2P extérieurs à la collaboration ou sur des agents de messages.

Le stockage des opérations chiffrées chez des tiers est une fonctionnalité qui améliorerait l'utilisabilité de l'application sans sacrifier la confidentialité des données. En effet, elle permettrait à un noeud déconnecté de manière temporaire de récupérer à sa reconnexion les modifications effectuées entretemps par ses collaborateur-rices, même si celleux-ci se sont depuis déconnecté-es.

Cependant, la clé de chiffrement est modifiée à la déconnexion du noeud. Ainsi, les opérations suivantes sont chiffrées avec une nouvelle clé que le noeud déconnecté ne possède pas. À sa reconnexion, ce dernier ne sera pas en mesure de déchiffrer et d'intégrer les opérations effectuées en son absence.

L'évolution de la clé de chiffrement de groupe à chaque connexion ou déconnexion d'un noeud est donc incompatible avec l'utilisation de cette même clé pour le stockage sécurisé des opérations chez des tiers. Une autre clé de chiffrement, dédiée, devrait donc être mise en place.

Une seconde limite liée à ce facteur d'évolution est la complexité en temps du protocole de génération de la clé de groupe. En effet, nos évaluations ont montré que ce protocole met jusqu'à 6 secondes pour s'exécuter.

Dans le cadre d'un système P2P à large échelle sujet au churn, des périodes d'activités où des noeuds se connectent et déconnectent toutes les 6 secondes sont à envisager. Lors de tels pics d'activités, les noeuds seraient incapables de collaborer, faute de clé de chiffrement de groupe. Il convient donc soit d'étudier l'utilisation d'autres protocoles de génération de clés de chiffrement de groupe plus efficaces, soit de considérer relaxer les garanties de *backward secrecy* et de *forward secrecy* dans le cadre des collaborations à large échelle.

2.7 Conclusion

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

MUTE permet d'éditer de manière collaborative des documents texte. Pour représenter les documents, MUTE propose plusieurs CRDTs pour le type Séquence [23, 24, 25] issus des travaux de l'équipe. 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 repose sur la technologie Web Real-Time Communication (WebRTC). Cette technologie permet de construire un réseau P2P directement entre plusieurs navigateurs. Plusieurs serveurs sont néanmoins requis, notamment pour la découverte des pairs et pour la communication entre des noeuds lorsque leur pare-feux respectifs empêchent 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 repose sur une clé de groupe de chiffrement qui est établie à l'aide du protocole [31].

Ce protocole nécessite que chaque noeud possède une paire de clés de chiffrement et qu'ils partagent leur clé publique. Pour partager leur clé publique, les noeuds utilisent des Public Key Infrastructures (PKIs). Cependant, afin de détecter un éventuel comportement malicieux de la part de ces derniers, MUTE intègre un mécanisme d'audit [26, 27].

Plusieurs tâches restent néanmoins à réaliser afin de répondre à notre objectif initial, c.-à-d. la conception d'un éditeur collaboratif P2P temps réel, à large échelle, sûr et sans autorités centrales. Une première d'entre elles concerne les droits d'accès aux documents. Actuellement, tout-e utilisateur-rice possédant l'URL d'un document peut découvrir les noeuds travaillant sur ce document, établir une connexion avec eux, participer à la génération d'une nouvelle clé de chiffrement de groupe puis obtenir l'état du document en se synchronisant avec un noeuds. Pour empêcher un tier ou un-e collaborateur-rice expulsé-e précédemment d'accéder au document, il est nécessaire d'intégrer un mécanisme de gestion de droits d'accès adapté aux systèmes P2P à large échelle [49, 50].

Une seconde piste de travail concerne l'amélioration de la couche réseau de MUTE. Comme mentionné précédemment (cf. sous-section 2.5.2, page 35), notre couche réseau actuelle souffre de plusieurs limites. Tout d'abord, les noeuds qui travaillent sur un même document établissent un réseau P2P entièrement maillé. Cette topologie réseau requiert un nombre de connexions par noeud qui dépend linéairement du nombre total de noeuds. Cette topologie s'avère donc coûteuse et inadaptée aux collaborations à large échelle.

Ensuite, le protocole de diffusion des messages que nous employons s'avère lui aussi inadapté aux collaborations à large échelle. En effet, c'est le noeud auteur d'un message qui a pour responsabilité d'en envoyer une copie à chacun des noeuds de la collaboration. Ainsi, ce protocole concentre toute la charge de travail pour la diffusion d'un message sur un seul noeud. Cette tâche s'avère excessive pour un unique noeud quand la collaboration est à large échelle.

Ces limites entravent donc la capacité à utiliser MUTE dans le cadre de collaborations à large échelle. Pour y répondre, nous identifions deux pistes d'amélioration :

(i) L'utilisation d'un protocole d'échantillonnage aléatoire de pairs adaptatif, tel que Spray [30], qui limite la taille du voisinage d'un noeud à un facteur logarithmique du nombre total de noeuds.

(ii) L'emploi d'un protocole de diffusion épidémique des messages [51], qui répartit sur l'ensemble des noeuds la charge de travail pour diffuser d'un message à tout le réseau.

Chapitre 3

Conclusions et perspectives

Sommaire

Dans ce chapitre, nous revenons sur les contributions présentées dans cette thèse. Nous rappelons le contexte dans lequel elles s'inscrivent, récapitulons leurs spécificités et apports, et finalement précisons plusieurs de leurs limites. Puis, nous concluons ce manuscrit en présentant plusieurs pistes de recherche qui nous restent à explorer à l'issue de cette thèse. La première s'inscrit dans la continuité directe de nos travaux sur un mécanisme de ré-identification pour CRDTs pour le type Séquence pour les applications Local-First Software (LFS). 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) [19] 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 copies des noeuds divergent alors temporairement. Les noeuds se synchronisent ensuite pour s'échanger leurs modifications

respectives. Les CRDTs garantissent alors la convergence forte à terme [19], c.-à-d. que les noeuds obtiennent de nouveau des copies équivalentes de la donnée.

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 [52].

Depuis leur formalisation, de nombreux travaux ont abouti à la conception de nouveaux CRDTs, soit en spécifiant de nouvelles sémantiques de résolution de conflits pour un type de données [53], soit en spécifiant de nouveaux modèles de synchronisation [54] ou en enrichissant les spécifications des modèles existants [55, 56].

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'alors 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 3.2.2.

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 le type Séquence

Pour privilégier leur disponibilité, latence et tolérance aux pannes, les systèmes distribués peuvent adopter le paradigme de la réplication optimiste [13]. 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 [18, 19] propose d'utiliser des types de données dont les modifications commutent nativement.

Depuis la spécification des CRDTs, la littérature a proposé plusieurs de ces mécanismes résolution de conflits automatiques pour le type de données Séquence [57, 58, 59, 60]. Cependant, ces approches souffrent toutes d'un surcoût croissant de manière monotone de leurs métadonnées et calculs. Ce problème a été identifié par la communauté, et celle-ci a proposé pour y répondre des mécanismes permettant soit de réduire la croissance de leur surcoût [61, 62], soit d'effectuer une Garbage Collection (GC) de leur surcoût [58, 20, 21]. 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 le type 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 [20, 21] : 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 [23].

Notre proposition prend la forme d'un nouveau CRDT pour le type Séquence à granularité variable : RenamableLogootSplit. Ce nouveau CRDT associe à LogootSplit un nouveau type de modification, *ren*, permettant de produire un nouvel état de la séquence équivalent 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, c.-à-d. le premier et le dernier des identifiants composant l'intervalle.

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. 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 générées par simulations.

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 le type 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 le type Séquence à granularité variable, e.g. un CRDT le type 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. Une 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 nos perspectives, c.-à-d. 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 [58, 55, 63].

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 [32, 33].

Cependant, les applications collaboratives sont historiquement des applications centralisées, e.g. Google Docs [35]. 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 applications Local-First Software (LFS) [12]. 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 Proof of Concept (PoC) pour les LFS. Ainsi, MUTE propose au moment où nous écrivons ces lignes un aperçu des travaux de recherche existants concernant :

(i) Les mécanismes de résolution de résolutions de conflits automatiques pour l'édition collaborative de documents textes [23, 24, 25].

(ii) Les protocoles distribués d'appartenance au groupe [28].

(iii) Les mécanismes d'anti-entropie [48].

(iv) Les protocoles distribués d'authentification d'utilisateur-rices [26, 27].

(v) Les protocoles distribués d'établissement de clés de chiffrement de groupe [31].

(vi) Les mécanismes de conscience de groupe.

Dans cette liste, nous avons personnellement contribué à l'implémentation des CRDTs LogootSplit [23] et RenamableLogootSplit [25], et du protocole d'appartenance au groupe SWIM [28].

En son état actuel, MUTE présente cependant 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 Web Real-Time Communication (WebRTC) repose sur l'utilisation de serveurs de signalisation, c.-à-d. de points de rendez-vous des pairs, et de serveurs de relais, 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 [64].

Dans le cadre de MUTE, nous avons opté pour cette seconde solution. Cependant, ce choix introduit un Single Point Of Failure (SPOF) [14] dans MUTE : OpenRelay elle-même. Afin de garantir la pérennité de MUTE, nous devrions reposer non pas sur une unique infrastructure de serveurs de signalisation et de relais mais sur une multitude. Malheureusement, l'écosystème actuel brille par la rareté d'infrastructures publiques offrant ces

14. Single Point Of Failure (SPOF) : Point de défaillance unique

services. Nous devons donc encourager et supporter la mise en place de telles infrastructures par une pluralité d'organisations.

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, plus aucun pair possédant une copie n'était disponible pour se synchroniser.

Notre collaborateur était donc dans l'incapacité de récupérer le document 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) [65], 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 limiterait ainsi le risque qu'un pair ne puisse récupérer l'état du document faute de pairs disponibles. Pour garantir l'utilisabilité du système P2P, une telle solution devrait donc permettre à un pair de récupérer l'état du document à sa reconnexion, malgré la potentielle évolution du groupe des collaborateur-rices et des pairs de stockage, e.g. l'ajout, l'éviction ou la déconnexion d'un des pairs. Cependant, la solution devrait en parallèle garantir qu'elle n'introduit aucune vulnérabilité, 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, e.g. si l'état courant comporte majoritairement des éléments ayant été insérés en concurrence de la génération de l'époque cible. 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 mesurer

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 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 [66]. 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'alors 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 la fusion de toute paire d'états possible en calculant leur borne supérieure, c.-à-d. leur Least Upper Bound (LUB) [67].

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 [56], 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 [68]. Dans de tels systèmes, les adversaires byzantins peuvent par exemple 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. [63] propose une solution adaptée aux systèmes P2P à large échelle. Celle-ci se base notamment sur l'utilisation de journaux infalsifiables. Il convient alors d'étudier si l'utilisation de ces 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.

3.2.3 Proposition d'un framework pour la conception de CRDTs synchronisés par opérations

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 [19, 56].

(ii) L'utilisation d'un framework [55] pour la conception de CRDTs purs synchronisés par opérations.

Malgré ses améliorations [69, 70], le framework présenté dans [55] souffre de plusieurs limitations, ce qui entrave la conception de nouveaux CRDTs synchronisés par opérations.

Tout d'abord, il ne permet que la conception de CRDTs purs synchronisés par opérations, c.-à-d. des CRDTs dont les modifications enrichies de leurs arguments et d'une estampille fournie par la couche de livraison des messages sont commutatives. Cette contrainte limite à des types de données simples, e.g. le Compteur ou l'Ensemble, les

CRDTs pouvant être spécifiés et exclut des types de données plus complexes, e.g. la Séquence ou le Graphe.

Une seconde limite de ce framework est qu'il repose sur le modèle de livraison causal des opérations. Ce modèle induit l'ajout de données de causalité précises à chaque opération, sous la forme d'un vecteur de version [71, 72] ou d'une barrière causale [73]. Nous jugeons ce modèle trop coûteux pour les applications LFS à large échelle.

Nous souhaitons donc proposer un nouveau framework pour la conception de CRDTs synchronisés par opérations répondant à ces limites. Nos objectifs sont multiples.

Notre framework devrait permettre la conception de CRDTs non-purs. Ce framework devrait aussi mettre en lumière la présence et le rôle de deux modèles de livraison dans les CRDTs synchronisés par opérations, ainsi que leur relation :

(i) Le modèle de livraison minimal requis par le CRDT pour assurer la convergence forte à terme [19].

(ii) Le modèle de livraison employé par le système qui utilise le CRDT, qui est une stratégie offrant un compromis entre modèle de cohérence et performances. Ce second modèle de livraison doit être égal ou plus contraint que le modèle de livraison minimal du CRDT.

Notamment, nous souhaitons expliciter que pour un CRDT synchronisé par opérations, le premier modèle de livraison est immuable tandis que le second peut être amené à évoluer en fonction de l'état du système et de ses besoins.

Par exemple, un système peut initialement garantir le modèle de cohérence causale [15] à ses utilisateur-rices, et pour cela utiliser le modèle de livraison causal. Cependant, ce modèle de livraison implique un surcoût qui dépend du nombre de noeuds du système. Ce modèle de livraison peut donc devenir trop coûteux si le nombre de noeuds devient important. Ainsi, le système peut décider de temporairement relaxer le modèle de cohérence garanti, e.g. garantir seulement le modèle de cohérence PRAM [16] [74], dans le but d'utiliser un modèle de livraison moins coûteux, e.g. le modèle de livraison FIFO [17].

15. Le modèle de cohérence causale est le modèle de cohérence auquel les utilisateur-rices sont généralement habitué-es : il garantit que l'ensemble des modifications seront intégrées dans leur ordre de génération (cf. **??**, page **??**).

16. Le modèle de cohérence PRAM garantit seulement que les modifications d'un noeud seront intégrés par les autres noeuds dans leur ordre de génération.

17. Le modèle de livraison FIFO garantit que les messages d'un noeud seront livrés aux autres noeuds dans le même ordre qu'ils ont été envoyés.

Annexe A

Liste des publications

Notre travail sur RenamableLogootSplit (cf. **??**, page **??**), 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 [75], nous motivons le problème identifié et présentons l'idée de RenamableLogootSplit, un CRDT pour le type Séquence incorporant un mécanisme de renommage.

(ii) Dans [76], nous détaillons une première version de RenamableLogootSplit et son mécanisme de renommage. Nous présentons dans cette version le mécanisme de résolution de conflits automatique conçu pour intégrer les opérations *ins* et *rmv* concurrentes aux opérations *ren*, et inversement. Nous présentons aussi dans ce travail notre protocole d'évaluation expérimentale ainsi que ses premiers résultats concernant l'impact du mécanisme de renommage sur le surcoût en métadonnées et en calculs du CRDT.

(iii) Dans [25], nous détaillons RenamableLogootSplit dans son entièreté. Ainsi, nous présentons les mécanismes additionnels pour :

  (i) Intégrer les opérations *ren* concurrentes.

  (ii) Supprimer à terme les métadonnées introduites par le mécanisme de renommage.

  Nous accompagnons cette proposition d'une évaluation expérimentale plus poussée, prenant en compte des scénarios avec des opérations *ren* concurrentes et mesurant de nouvelles métriques telles que le temps nécessaire pour rejouer le journal des opérations d'une collaboration. Finalement, nous complétons notre travail d'une discussion identifiant plusieurs des limites de RenamableLogootSplit et présentant des pistes possibles pour y répondre.

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

Efficient renaming in CRDTs [75]

Auteur Matthieu Nicolas

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

Résumé *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 [76]

Auteurs Matthieu Nicolas, Gérald Oster, Olivier Perrin

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

Résumé 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 [25]

Auteurs Matthieu Nicolas, Gérald Oster, Olivier Perrin

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

Résumé 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.

Annexe B

Entrelacement d'insertions concurrentes dans Treedoc

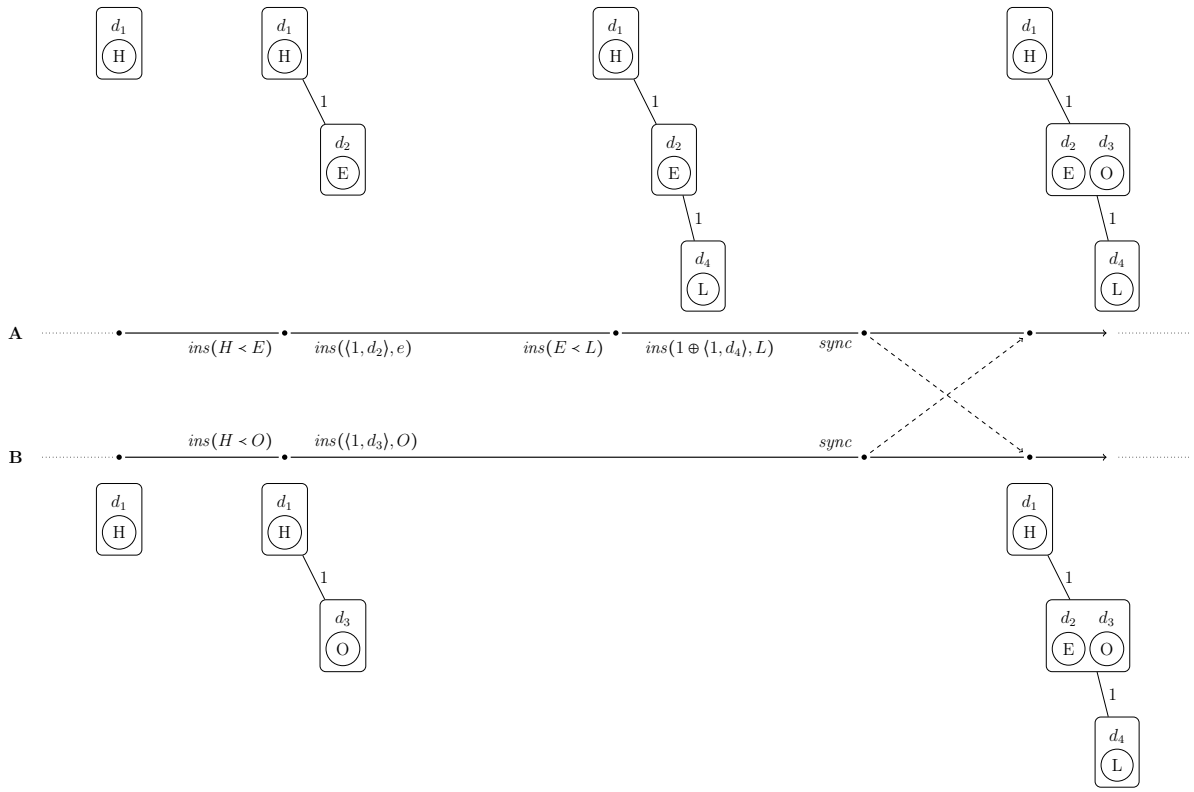


FIGURE B.1 – Entrelacement d'insertions concurrentes dans une séquence Treedoc

Annexe C

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 D

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
```

Bibliographie

[1] Ina Fried. *Scoop : Google's G Suite cracks 2 billion users*. Last Accessed : 2022-10-19. URL : https://www.axios.com/2020/03/12/google-g-suite-total-users.

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

[3] 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/.

[4] GitHub. *Search · type :user*. Last Accessed : 2022-10-19. URL : https://github.com/search?q=type:user&type=Users.

[5] Taylor Lorenz. *Internet communities are battling over pixels*. Last Accessed : 2022-10-18. URL : https://www.washingtonpost.com/technology/2022/04/04/reddit-place-internet-communities/.

[6] Matthew O'Mara. *Twitch Plays Pokémon a wild experiment in crowd sourced gameplay*. Last Accessed : 2022-10-18. URL : https://financialpost.com/technology/gaming/twitch-plays-pokemon-a-wild-experiment-in-crowd-sourced-gameplay.

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

[8] Wikipedia. *Censorship of Wikipedia*. Last Accessed : 2022-10-18. URL : https://en.wikipedia.org/wiki/Censorship_of_Wikipedia.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

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

[43] 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.

[44] John GRUBER. *Daring Fireball : Markdown*. Last Accessed : 2022-10-17. URL : https://daringfireball.net/projects/markdown/.

[45] Geoffrey LITT, Sarah LIM, Martin KLEPPMANN et Peter van HARDENBERG. « Peritext : A CRDT for Collaborative Rich Text Editing ». In : *Proceedings of the ACM on Human-Computer Interaction (PACMHCI)* 6.MHCI (nov. 2022). DOI : 10.1145/3555644. URL : https://doi.org/10.1145/3555644.

[46] 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.

[47] 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.

[48] 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.

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

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

[51] Kenneth P BIRMAN, Mark HAYDEN, Oznur OZKASAP, Zhen XIAO, Mihai BUDIU et Yaron MINSKY. « Bimodal multicast ». In : *ACM Transactions on Computer Systems (TOCS)* 17.2 (1999), p. 41–88.

[52] 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.

[53] 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.

[54] 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.

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

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

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

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

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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.

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

[72] 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.

[73] 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.

[74] Richard J Lipton et Jonathan S Sandberg. *PRAM : A scalable shared memory*. Princeton University, Department of Computer Science, 1988.

[75] 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.

[76] 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.

Résumé

Un système collaboratif permet à plusieurs utilisateur-rices de créer ensemble un contenu. Afin de supporter des collaborations impliquant des millions d'utilisateurs, ces systèmes adoptent une architecture décentralisée pour garantir leur haute disponibilité, tolérance aux pannes et capacité de passage à l'échelle. Cependant, ces sytèmes échouent à garantir la confidentialité des données, souveraineté des données, pérennité et résistance à la censure. Pour répondre à ce problème, la littérature propose la conception d'applications Local-First Software (LFS) : des applications collaboratives pair-à-pair (P2P).

Une pierre angulaire des applications LFS sont les Conflict-free Replicated Data Types (CRDTs). Il s'agit de nouvelles spécifications des types de données, tels que l'Ensemble ou la Séquence, permettant à un ensemble de noeuds de répliquer une donnée. Les CRDTs permettent aux noeuds de consulter et de modifier la donnée sans coordination préalable, et incorporent un mécanisme de résolution de conflits pour intégrer les modifications concurrentes. Cependant, les CRDTs pour le type Séquence souffrent d'une croissance monotone du surcoût de leur mécanisme de résolution de conflits. Pouvons-nous proposer un mécanisme de réduction du surcoût des CRDTs pour le type Séquence qui soit compatible avec les applications LFS ? Dans cette thèse, nous proposons un nouveau CRDT pour le type Séquence, RenamableLogootSplit. Ce CRDT intègre un mécanisme de renommage qui minimise périodiquement le surcoût de son mécanisme de résolution de conflits ainsi qu'un mécanisme de résolution de conflits pour intégrer les modifications concurrentes à un renommage. Finalement, nous proposons un mécanisme de Garbage Collection (GC) qui supprime à terme le propre surcoût du mécanisme de renommage.

Abstract

A collaborative system enables multiple users to work together to create content. To support collaborations involving millions of users, these systems adopt a decentralised architecture to ensure high availability, fault tolerance and scalability. However, these systems fail to guarantee the data confidentiality, data sovereignty, longevity and resistance to censorship. To address this problem, the literature proposes the design of Local-First Software (LFS) applications : collaborative peer-to-peer applications.

A cornerstone of LFS applications are Conflict-free Replicated Data Types (CRDTs). CRDTs are new specifications of data types, e.g. Set or Sequence, enabling a set of nodes to replicate a data. CRDTs enable nodes to access and modify the data without prior coordination, and incorporate a conflict resolution mechanism to integrate concurrent modifications. However, Sequence CRDTs suffer from a monotonous growth in the overhead of their conflict resolution mechanism. Can we propose a mechanism for reducing the overhead of Sequence-type CRDTs that is compatible with LFS applications ? In this thesis, we propose a novel CRDT for the Sequence type, RenamableLogootSplit. This CRDT embeds a renaming mechanism that periodically minimizes the overhead of its conflict resolution mechanism as well as a conflict resolution mechanism to integrate concurrent modifications to a rename. Finally, we propose a mechanism of Garbage Collection (GC) that eventually removes the own overhead of the renaming mechanism.